

N 69 2368 3
NASA CR 00667

ELECTRICAL

THE DESIGN AND IMPLEMENTATION OF A LABORATORY-ORIENTED
GENERALIZED VARIABLE-ORDER DIGITAL COMPENSATOR

PREPARED BY

DIGITAL SYSTEMS LABORATORY

AUBURN UNIVERSITY

C. C. Carroll, Project Leader

Eleventh Technical Report

May, 1968

CONTRACT NAS8-11274
GEORGE C. MARSHALL SPACE FLIGHT CENTER
NATIONAL AERONAUTICS AND SPACE ADMINISTRATION
HUNTSVILLE, ALABAMA

ENGINEERING EXPERIMENT STATION

AUBURN UNIVERSITY

AUBURN, ALABAMA

E
N
G
I
N
E
E
R
I
N
G

THE DESIGN AND IMPLEMENTATION OF A LABORATORY-ORIENTED
GENERALIZED VARIABLE-ORDER DIGITAL COMPENSATOR

PREPARED BY

DIGITAL SYSTEMS LABORATORY

AUBURN UNIVERSITY

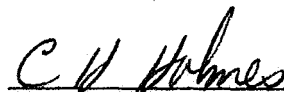
C. C. Carroll, Project Leader

Eleventh Technical Report

May, 1968

CONTRACT NAS8-11274
GEORGE C. MARSHALL SPACE FLIGHT CENTER
NATIONAL AERONAUTICS AND SPACE ADMINISTRATION
HUNTSVILLE, ALABAMA

APPROVED BY:



C. H. Holmes
Head Professor
Electrical Engineering

SUBMITTED BY:



C. C. Carroll
Associate Professor
Electrical Engineering

FOREWORD

This document is a technical summary of the progress made since January 28, 1967, by the Auburn University Electrical Engineering Department toward fulfillment of Phase B of Contract No. NAS8-11274. This contract was awarded to Auburn Research Foundation, Auburn, Alabama, May 28, 1964, and was extended September 28, 1966 by the George C. Marshall Space Flight Center, National Aeronautics and Space Administration, Huntsville, Alabama.

ACKNOWLEDGMENTS

The authors wish to express their appreciation to Hubert T. Nagle for his valuable assistance with the software design. Bob Heath and George Jordan were also instrumental in documenting and clarifying the final design.

Finally, the authors would like to thank Ralph Cavin and John C. Johnson for their cooperation in furnishing and interfacing the computer program for the flight dynamics of the Saturn V rocket with the computer program for the compensator.

THE DESIGN AND IMPLEMENTATION OF A LABORATORY-ORIENTED GENERALIZED VARIABLE-ORDER DIGITAL COMPENSATOR

C. C. Carroll and J. A. Childs

ABSTRACT

This study is concerned with the design and implementation of a versatile digital compensator that is compatible with a laboratory environment. The design permits future construction to be facilitated almost completely through the use of integrated circuitry.

The versatility of the compensator is demonstrated in two ways:

- 1) the compensation function can be altered to give 2nd, 4th, or 6th order operation by making minor external adjustments to the machine and
- 2) for a given order, the coefficients of the transfer function can be set to a wide range of values, thereby providing the capability of an even greater variety of realizable functions. The resolution of the coefficients has been made high enough to eliminate coefficient quantization as a problem source.

The compensator, as a special purpose computer, will be utilized in the hybrid simulation of the Saturn V thrust vector control system, although the computer may be adapted to many diverse sampled-data systems.

TABLE OF CONTENTS

LIST OF FIGURES	vii
I. INTRODUCTION	1
II. DESIGN OF THE COMPENSATOR	4
III. IMPLEMENTATION OF THE COMPENSATOR	17
A. Introduction	17
B. Parallel Binary Accumulator	17
C. Shift Register	20
D. Storage Registers	24
E. Address Command Processor	26
F. Multiplication Logic	29
G. Coefficient Switches	31
H. Truncation Logic	31
I. 'A' and 'B' Memories.	36
J. Input Select Logic	36
K. Manual Select Logic	39
L. Master Controller	39
IV. THEORETICAL RESULTS	49
A. System Response	49
B. Sampling Frequency and Delay Considerations	52
V. USE OF THE COMPENSATOR AS A PIECE OF LABORATORY EQUIPMENT . .	54
VI. CONCLUSIONS	57

REFERENCES	58
APPENDICES	60
A. Fortran Source Program for Simulating the Flight Dynamics and Compensator	61
B. Fortran Source Program for Determining the Positioning of the Coefficient Switches	74

LIST OF FIGURES

1. Block Diagram of Flight Simulation	2
2. Block Diagram of $D(z)$	3
3. Sampling of the Analog Input	6
4. Timing Sequence for Computation Scheme	10
5. Mathematical Model of Cascaded Compensators	11
6. Component Block Diagram of Compensator	13
7. Logic Diagram of Parallel Binary Accumulator	19
8. Typical Shift Register Configuration	21
9. Transition Table for the Flip-Flops	21
10. Logic Diagram for Shift Register	23
11. Typical Storage Register Array	25
12. Clocking Scheme for Storage Registers	27
13. Address Command Processor	28
14. Multiplication Logic	30
15. Coefficient Switches	32
16. Typical Truncation Logic	34
17. Logic Diagram for Both 'A' and 'B' Memories	37
18. Input Select Logic	38
19. Manual Select Logic	40
20. Logic Diagram of Master Controller	41
21. Timing Pulses for Master Controller Delay-Multivibrators	43

22.	Time Response for Fifth-Order Compensation Function	50
23.	Time Response for Sixth-Order Compensation Function	51
24.	Control Panel Layout	55

I. INTRODUCTION

Recent interest in digital filtering [5,6,7] has created a demand for physical realizations of z-plane transfer functions; and, as a result, physical realizations have been accomplished through the use of several methods [8,9,10,11,12,13]. The design of a special purpose computer for realizing a digital filter consists of three parts [14]: first, the determination of quantization levels in the computer; second, the logical design of the computer's components; and third, the interconnection of the computer's components to implement the required quantization levels. In this paper, a special purpose computer is designed using the above concepts to realize a given $D(z)$ transfer function of a digital filter. The digital filter is then used as a compensator in the hybrid simulation of the Saturn V thrust vector control system shown in Figure 1. The compensator will be of the general form shown in Figure 2 with the following salient characteristics: 1) the order of the $D(z)$ may be altered to give 2nd, 4th, or 6th order capability by making minor external adjustments and 2) for a given order, the coefficients of the transfer function can be set to a wide range of values, thereby providing the capability of an even greater variety of realizable functions.

Chapter II describes the design problem, while Chapter III details the actual logic implementation of each computer component used in the design. In Chapter IV, the theoretical performance of the machine is analyzed. Chapter V presents a step-by-step procedure for programming the computer for a desired transfer function.

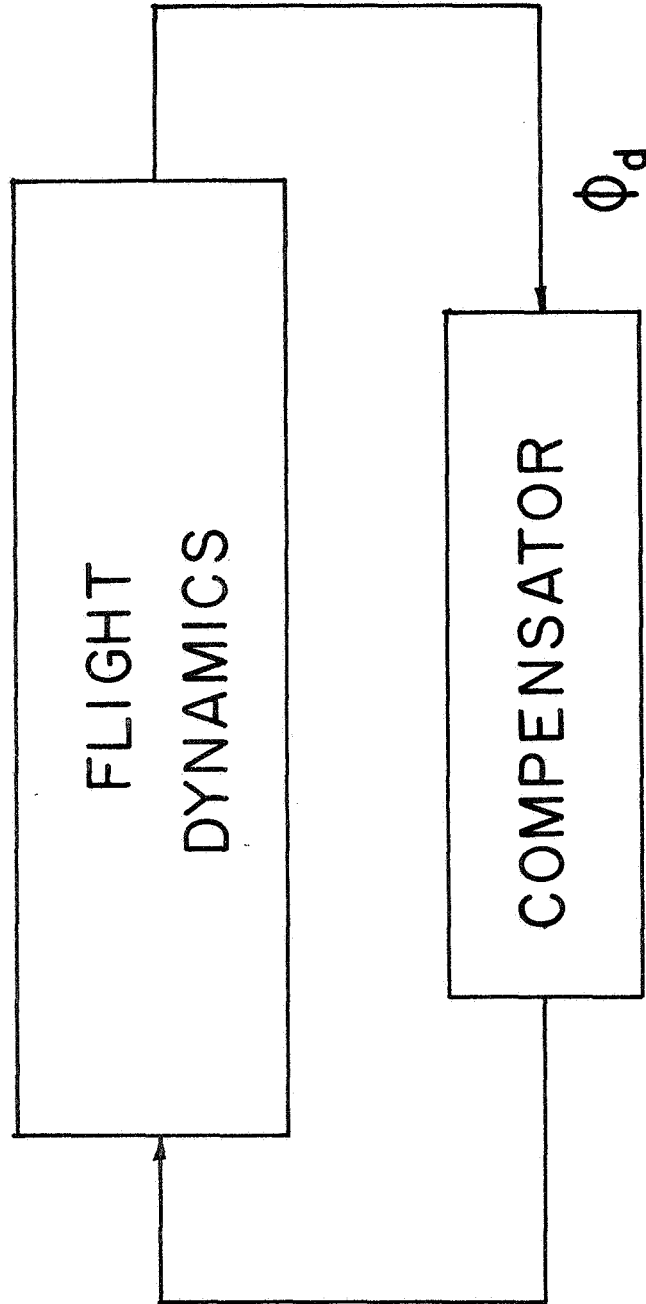


Fig. 1--Block Diagram of Flight Simulation

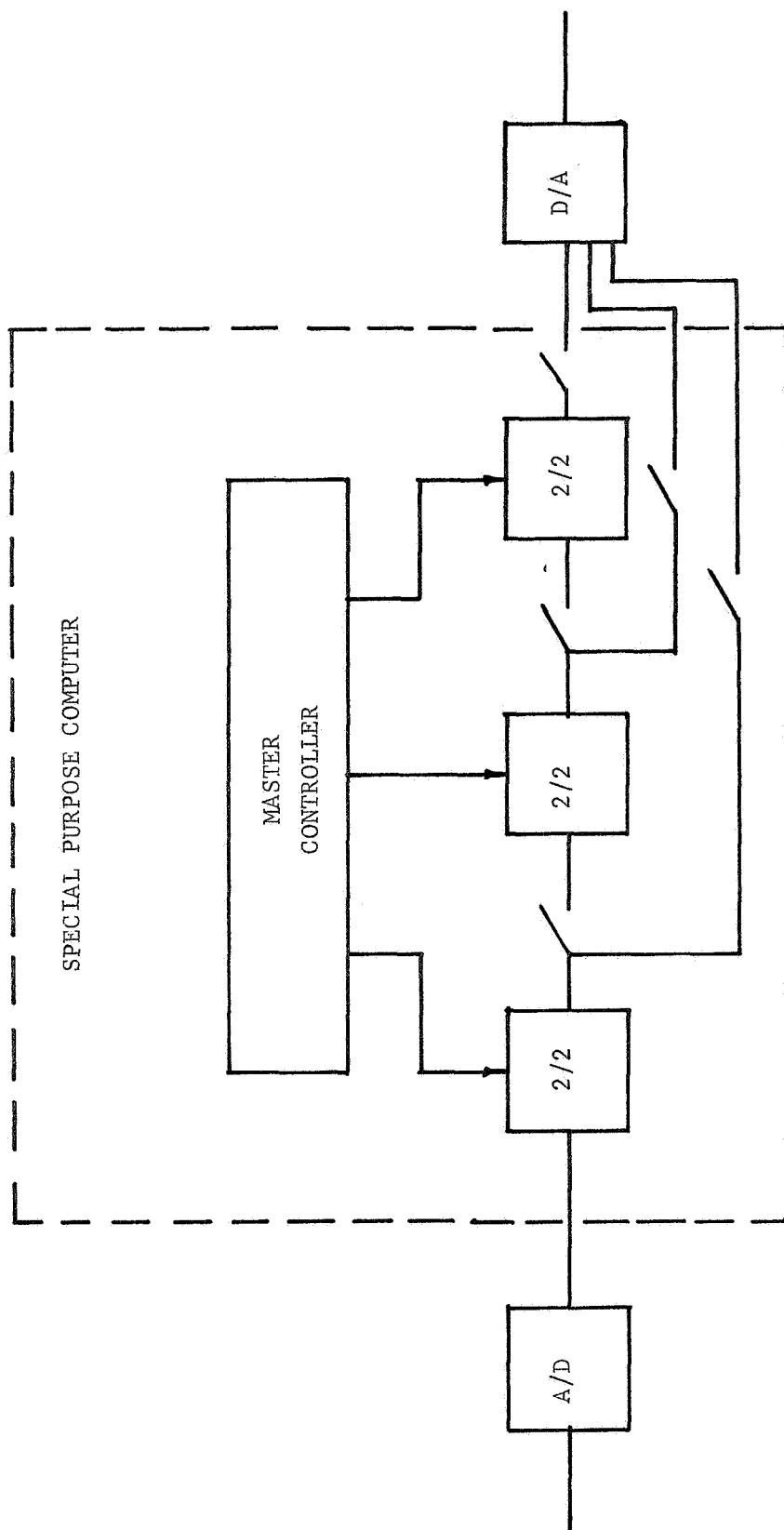


Fig. 2 --Block Diagram of $D(z)$

II. DESIGN OF THE COMPENSATOR

The form of the $D(z)$ to be implemented is given below:

$$D(z) = \frac{a_0 z^2 + a_1 z + a_2}{z^2 + b_1 z + b_2} \times \frac{z^2 + a_1' z + a_2'}{z^2 + b_1' z + b_2'} \times \frac{z^2 + a_1'' z + a_2''}{z^2 + b_1'' z + b_2''} \quad (1)$$

The sixth-order function given above is factored into three second-order functions. Each of these three factors will be represented by a difference equation in order to facilitate the actual digital design. These three equations can then be combined in such a manner as to describe the complete sixth-order function. Also these equations may be arranged to represent second- and fourth-order functions. In addition, the coefficients of the $D(z)$ may be changed to yield entirely different transfer functions.

The sixth-order case for a $D(z)$ can perhaps be better understood by first analyzing the second-order transfer function given below:

$$D(z) = \frac{E_2(z)}{E_1(z)} = \frac{a_0 + a_1 z^{-1} + a_2 z^{-2}}{1 + b_1 z^{-1} + b_2 z^{-2}} \quad (2)$$

Expanding Eq. 2 yields

$$E_2(z)[1 + b_1 z^{-1} + b_2 z^{-2}] = E_1(z)[a_0 + a_1 z^{-1} + a_2 z^{-2}] \quad (3)$$

By taking the inverse z -transform of the above equation and rearranging

terms, the output in standard difference notation is

$$\begin{aligned}
 e_2(KT) = & a_0 e_1(KT) + a_1 e_1(KT-T) + a_2 e_1(KT-2T) \\
 & - b_1 e_2(KT-T) - b_2 e_2(KT-2T) \quad .
 \end{aligned} \tag{4}$$

Therefore, the present output, $e_2(KT)$, is a linear combination of two previous inputs, two previous outputs, and the present input, since $e_1(KT-2T)$ represents the value of the input signal two sampling periods ago as shown in Figure 3.

The difference equation for the second function of Eq. (1) is

$$\begin{aligned}
 e_2'(KT) = & e_1'(KT) + a_1' e_1'(KT-T) + a_2' e_1'(KT-2T) \\
 & - b_1' e_2'(KT-T) - b_2' e_2'(KT-2T)
 \end{aligned} \tag{5}$$

The difference equation for the third function of Eq. (1) and the final output is

$$\begin{aligned}
 e_2''(KT) = & e_1''(KT) + a_1'' e_1''(KT-T) + a_2'' e_1''(KT-2T) \\
 & - b_1'' e_2''(KT-T) - b_2'' e_2''(KT-2T)
 \end{aligned} \tag{6}$$

Now the problem of combining these three equations into one equation to describe all three stages of the $D(z)$ is encountered. Equations (4),

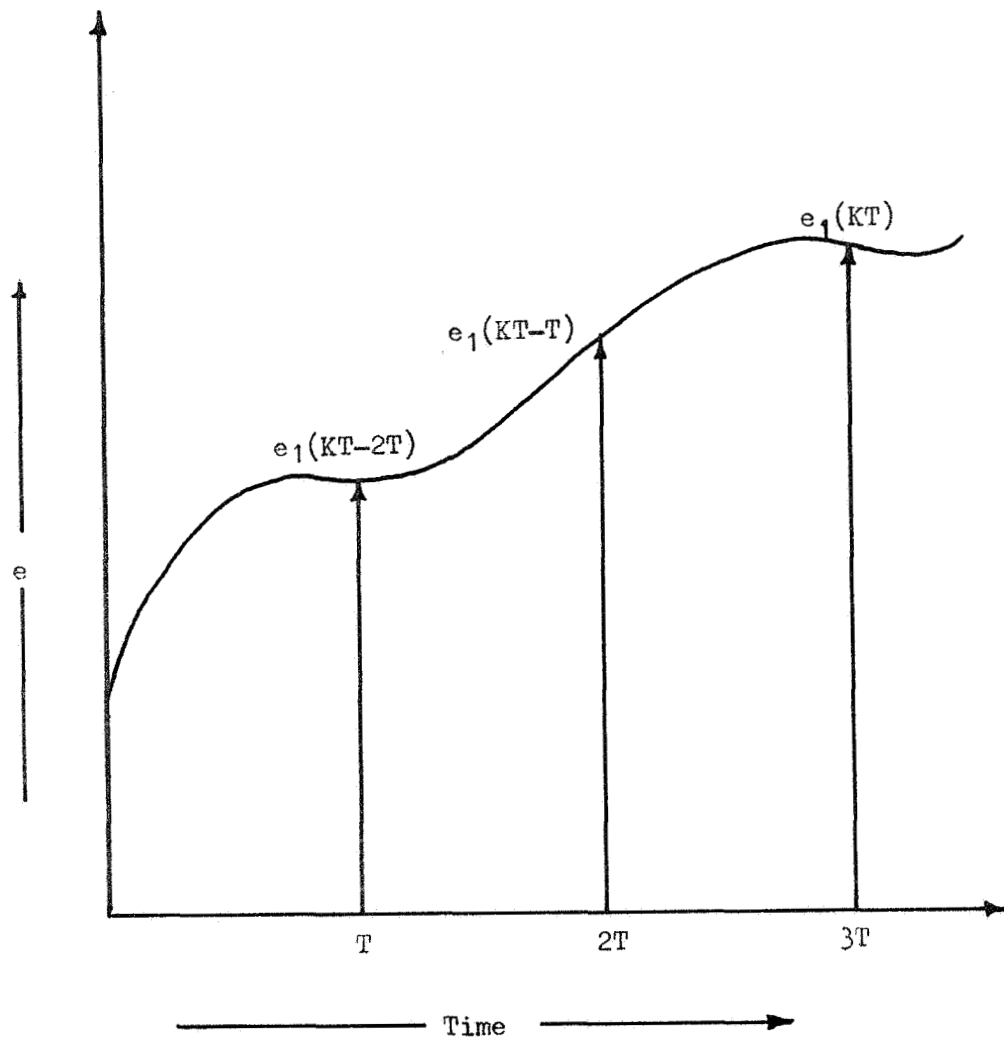


Fig. 3 - Sampling of the Analog Input

(5), and (6) can be combined to give the difference equation for Eq. (1), since

$$e_1'(KT) = e_2(KT) \quad (7)$$

$$e_1''(KT) = e_2'(KT) \quad (8)$$

Using the above relations the final output can be rewritten as:

$$\begin{aligned} e_2''(KT) = & a_o e_1(KT) + \overbrace{a_1 e_1(KT-T) + a_2 e_1(KT-2T) - b_1 e_2(KT-T) - b_2 e_2(KT-2T)}^W \\ & + \overbrace{a_1' e_1'(KT-T) + a_2' e_1'(KT-2T) - b_1' e_2'(KT-T) - b_2' e_2'(KT-2T)}^X \\ & + \overbrace{a_1'' e_1''(KT-T) + a_2'' e_1''(KT-2T) - b_1'' e_2''(KT-T) - b_2'' e_2''(KT-2T)}^Y \end{aligned} \quad (9)$$

or

$$e_2''(KT) = a_o e_1(KT) + W + X + Y \quad (10)$$

Equations (3), (4), and (5) can also be rewritten as:

$$e_2(KT) = a_o e_1(KT) + W \quad (11)$$

$$e_2'(KT) = e_1'(KT) + X \quad (12)$$

$$e''(KT) = e_1''(KT) + Y \quad (13)$$

As indicated before, Eq. (9) states that the present output is dependent on previous inputs and outputs and also on the present input,

$e_1(KT)$. In order to compute the difference equation output, $e_2''(KT)$, twelve additions and thirteen multiplications must be computed. It is desired to perform as many calculations as possible before the input is actually sampled since Eq. (9) implies that the output is available at the same instant of time that the input, $e_1(KT)$, is obtained by the system. Therefore a computation scheme is devised whereby all terms involving previous values of e_1 and e_2 are multiplied by their corresponding coefficients and summed before $e_1(KT)$ is obtained. When $e_1(KT)$ is obtained, it is multiplied by a_0 , and the product is added to the previously mentioned sum to give the present output, $e_2''(KT)$.

The scheme of computation will be as follows: X is computed and stored; Y is computed and stored; and W is computed and left in the accumulator. Then the input is sampled, multiplied by a_0 , and added to W in the accumulator which gives the value of the output, $e_2(KT)$, of the first stage. The previously stored value of X is added to $e_2(KT)$ to form the output, $e_2'(KT)$, of the second stage, or fourth order $D(z)$. Finally Y is added to the above total to give the output, $e_2''(KT)$, of the sixth order $D(z)$ or third stage. Perhaps the above computational procedure can be clarified somewhat by the following equations:

$$e_2(KT) = a_0 e_1(KT) + W \quad (14)$$

$$e_2'(KT) = a_0 e_1(KT) + W + X \quad (15)$$

$$e_2''(KT) = a_o e_1(KT) + W + X + Y . \quad (16)$$

The above procedures are illustrated in Figure 4 where T_s is the sampling period and T_{\min} corresponds to the minimum sampling period or the maximum sampling frequency. The time intervals T_x , T_y , and T_w represent the time sequence of the computation of X, Y, and W respectively. As mentioned before, it is desirable to have an output as soon as an input is obtained, but a certain amount of calculation time is always required. This calculation time is represented by the interval, T_c , in Figure 4. T_c then represents the delay time between input and output; therefore, its value should be made small. During the time interval T_c , $e_1(KT)$ is sampled, multiplied by a_o , and then added to either W, W+X, or W + X + Y depending upon the order of the compensator being used.

Now a mathematical model of the system which will clarify and integrate the preceding techniques can be presented. The model is shown in Figure 5, which depicts a separate summing network for each stage, but in reality one summer, or accumulator, is time-shared by all three stages. This results in savings of time, space, and money. Each triangle in this figure corresponds to the multiplication operation, and each square block represents a time delay element in which the previous values of the input and output are delayed and stored. Figure 5 is divided into three stages to represent the computation of W, X, and Y, respectively. It is important to note at this point that even if $e_2''(KT)$ is desired, the separate values of $e_2(KT)$ and $e_2'(KT)$ must be

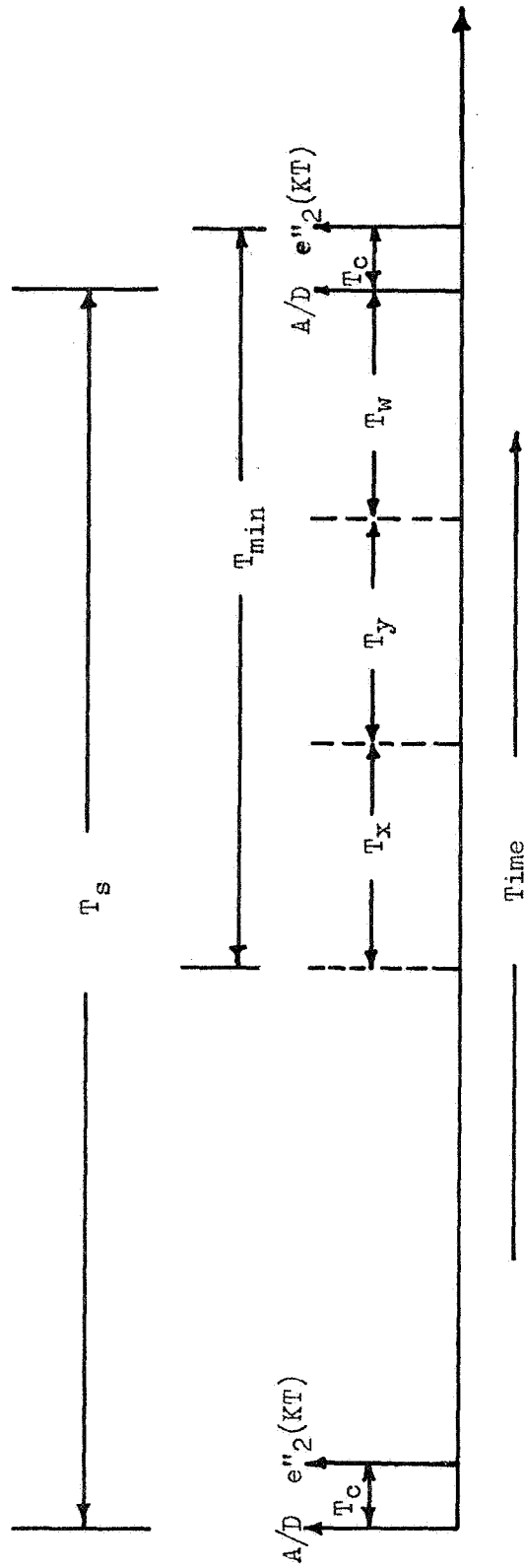


Fig. 4 Timing Sequence for Computation Scheme

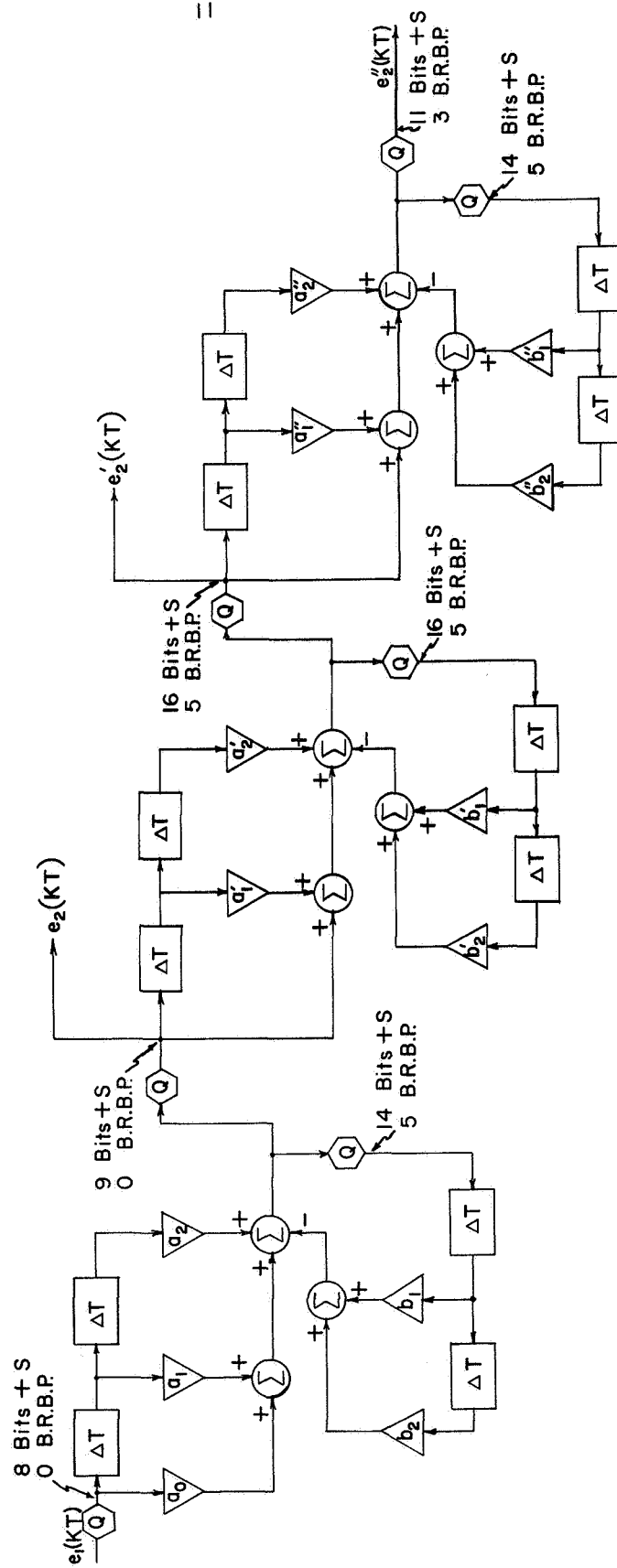


Fig. 5 -- Mathematical Model of Cascaded Compensators.

computed so that inputs to the second and third stages will be available when the next sampling action occurs.

Equation (9) indicates that several sets of digital numbers will have to be multiplied together, so the method of multiplication will be explained before proceeding to the block diagram of the complete system. Suppose, for instance, that $e_2(KT-T)$ equals $(1000.0)_2$ and it is desired to multiply this by b_1 which is equal to $(0.01)_2$. The product is obtained by shifting the binary point two places to the left to give $(10.0)_2$. The product in decimal is $(8)(1/4) = 2$. Three shifts to the left of the binary point would correspond to multiplication by $1/8$.

The following example will illustrate the above techniques. Suppose $b_1 e_2(KT-T) = (1.375)(8.0)$, then the multiplication would be carried out as below:

$$\begin{aligned}
 (1.375)(8) &= (1 + 1/4 + 1/8)(8) \\
 &= (1.011)_2 (1000)_2 = (1 + .01 + .001)_2 (1000)_2 \\
 &= (1011.0)_2 = (11)_{10}
 \end{aligned}$$

1000.0	no shift
10.00	2 shifts
1.000	3 shifts
(1011.000) ₂	= (11.0) ₁₀

Now that a background of the problem definition and computation scheme has been given, a block diagram for the design of the compensator is given in Figure 6. Referring again to Eq. (9), one can see that a digital design must be created whereby these binary

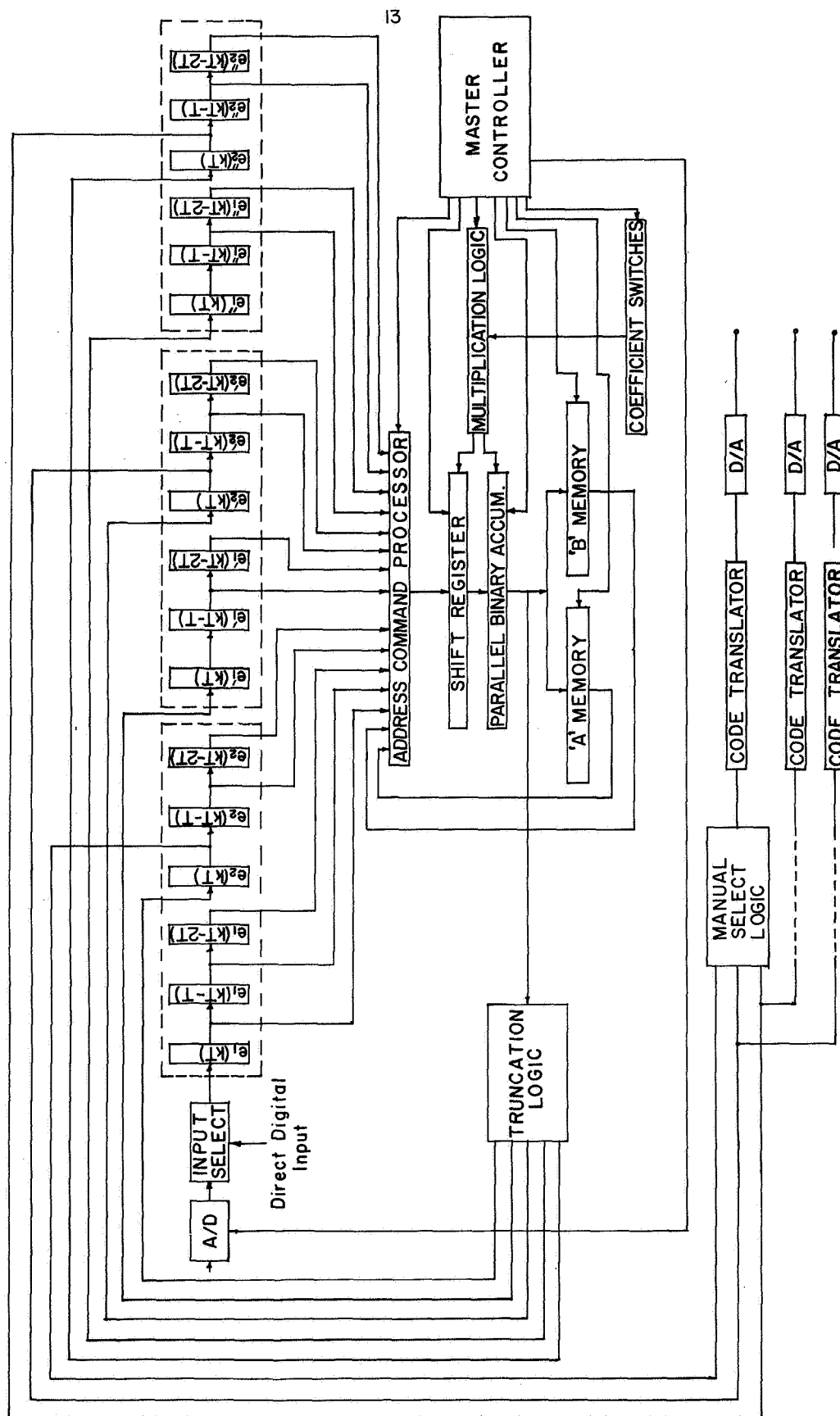


Fig. 6 ---Component Block Diagram of Compensator

numbers can be multiplied by their corresponding coefficients and then summed in the prescribed sequence to give an output.

The sequence of operation of the compensator will now be described. The machine is a special purpose computer and, as such, must be pre-programmed. The coefficient switches are set to give the desired coefficients which involves the positioning of about two hundred and thirteen switches. Next, the output, $e_2(KT)$, $e_2'(KT)$, or $e_2''(KT)$ is selected by the positioning of three switches, and the type of input, analog or digital, to be used is selected by the use of another switch. The clock frequency is adjusted to the desired sampling rate.

The master controller first shifts the following registers:

$e_1(KT-T)$, $e_1(KT-2T)$, $e_2(KT-T)$, $e_2(KT-2T)$, $e_1'(KT-T)$, $e_1'(KT-2T)$,
 $e_2'(KT-T)$, $e_2'(KT-2T)$, $e_1''(KT-T)$, $e_1''(KT-2T)$, $e_2''(KT-T)$, and $e_2''(KT-2T)$.

This is done to allow the calculation of X, Y, and W before the input is actually sampled. It should be noted at this point that only the storage registers listed above contain a new word. The master controller then instructs the address command processor to address the information in the $e_1'(KT-T)$ register and load $e_1'(KT)$ into the shift register.

The multiplication logic serves to shift the number in the shift register and also to control the assertion of the accumulate commands. An accumulate pulse transfers the information in the shift register into the parallel binary accumulator. The shifting pulses occur at regular intervals, but the occurrence of an accumulate pulse is predetermined by the settings on the coefficient switches. The

shifting and accumulating process performs the multiplication of a_1' by $e_1'(KT-T)$.

A similar procedure is carried out for $a_2'e_1'(KT-2T)$, $b_1'e_2'(KT-T)$, and $b_2'e_2'(KT-2T)$, after which the master controller commands the multiplication logic to go to the stand-by mode. During this idling period, the controller initiates a command whereby the information in the accumulator is shifted into the 'A' memory. This transferred information represents the value of X, as defined before. Once the accumulator has been interrogated, it is reset and simultaneously the controller prepares itself to continue the computations.

Next, the $e_1''(KT-T)$, $e_1''(KT-2T)$, $e_2''(KT-T)$, and $e_2''(KT-2T)$ storage registers are successively interrogated by the address command processor and multiplied by a_1'' , a_2'' , b_1'' , b_2'' respectively in a manner as prescribed before. Once again the multiplication logic is put in the stand-by mode, and the information in the accumulator, which now represents Y, is shifted into the 'B' memory. The accumulator is again reset with the value of Y stored in the 'B' memory and the value of X stored in the 'A' memory.

Finally $e_1(KT-T)$, $e_1(KT-2T)$, $e_2(KT-T)$, and $e_2(KT-2T)$ are processed and multiplied by their corresponding coefficients, a_1 , a_2 , b_1 , and b_2 . The value representing W is left in the accumulator, and is not transferred as X and Y had been. The master controller, having completed the above operations, automatically switches itself to the sample mode. The controller sends eight pulses to the successive approximation analog-to-digital converter [2], where

the value of the analog signal is sampled and converted to a binary number. After these eight A/D pulses have been applied, two end-of-A/D pulses are generated. The first pulse simultaneously shifts the information out of the A/D and into the $e_1(KT)$ register. The second pulse resets the A/D and also returns the master control logic to the computation mode. The address command processor is then given a command to interrogate the $e_1(KT)$ register. $e_1(KT)$ is then multiplied by a_0 and stored in the accumulator. The accumulator now contains the value of $e_2(KT)$, which will be channeled in two directions simultaneously. A pulse which will shift this stored information to the output of the compensator through one type of truncation or round-off logic will be actuated. This same information will also be truncated internally in the accumulator and stored in the $e_1'(KT)$ register.

The master controller then recalls the word stored in memory 'A' and applies it to the inputs of the accumulator. The accumulator now contains $e_2'(KT)$, which is routed both to the $e_1''(KT)$ register and to the output for a fourth-order machine. Y is now in the 'B' memory and it is added to the value of $e_1''(KT)$ presently in the accumulator to give the output, $e_2''(KT)$, for a sixth-order compensator. To complete the cycle and prepare the system for the next sampling period, the accumulator is reset and the master controller is switched to the stand-by mode.

The above explanation was meant to give only a general insight into the system operation. In the following chapters, a more detailed analysis will be presented in which some of the preceding explanations are repeated using a different approach.

III. IMPLEMENTATION OF THE COMPENSATOR

A. Introduction

Now that the system operation has been defined from a block diagram point of view, each of the blocks in Figure 6 will be explained with the exception of the analog-to-digital converter, the code translator logic, and the digital-to-analog converter. The parallel binary accumulator will be explained briefly. Most of the above elements have been used in previous designs [1], [2]; therefore, they will not be explained here.

In Chapter II the general approach was to describe the compensator components in the order in which they occurred in the sequence of operation. But in this chapter an attempt will be made to give the functional operation of each block in the order of design evolution. The combination of these two approaches should clarify matters considerably.

B. Parallel Binary Accumulator

The parallel binary accumulator is actually a parallel adder in that words are entered in parallel for the adding operation, but the accumulator, as its name implies, keeps a total of all numbers that are added or subtracted. The sign of the numbers that are entered into the machine is determined by a sign bit, which is located to the left of the most significant bit. For instance, 0-100 represents a positive four while 1-100 represents a negative four.

Data entry into the accumulator of Figure 7 is made through the use of points A, B, and C, while an output is obtained at the points D, E, and F. In order to accumulate a number, an accumulate pulse must be fed into point H. If it is desired to reset all stages of the accumulator, then a clear or reset pulse may be applied to the common reset terminal. In some instances it will be desirable to reset only certain stages of the accumulator by applying reset pulses to points F and G. The reasoning for this will be given in a later section on truncation.

A simplified version of the stages of the accumulator is presented later which shows the positioning of the binary point and the number of stages required to the left and to the right of this point. The determination of the number of stages to the left of the binary point was obtained from Eq. (9) by assuming that the coefficients had a maximum decimal value of approximately two. The decimal equivalents of the sampled values were multiplied by two and added together to give the maximum number that the accumulator would ever have to process. Next, the maximum number of bits required to the right of the binary point was determined by first finding the entry word having the most bits to the right of the binary point. The entry word, $e_2'(KT)$, has the most bits to the right of the binary point, 5, so five is added to sixteen to give the total number of stages to the right of the binary point. Most of the coefficients have seventeen bits, but one of these bits is to the left of the binary point. This is the reason that sixteen instead of seventeen was used in the above computations.

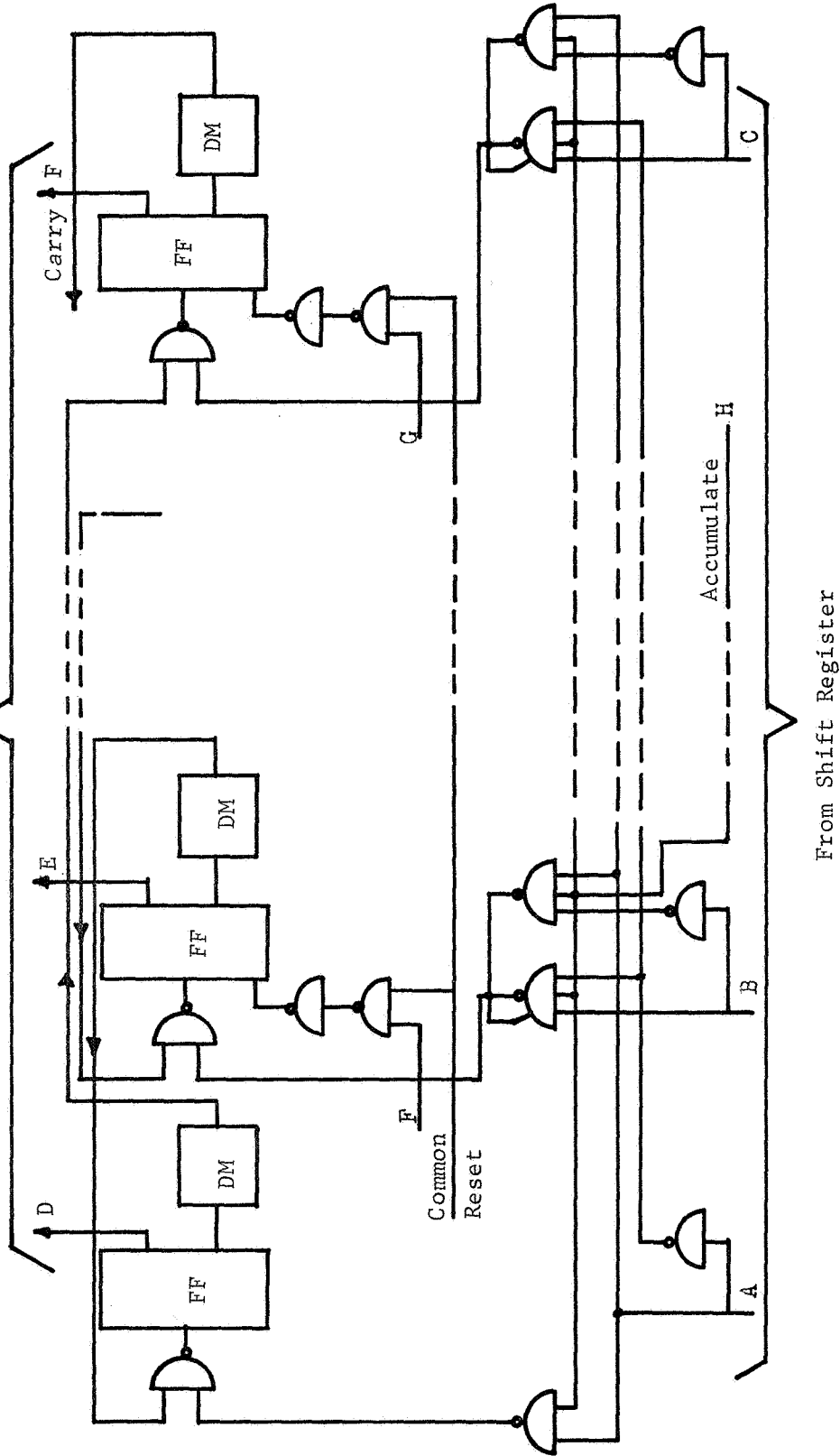


Fig. 7 --Logic Diagram of Parallel Binary Accumulator

Referring again to Figure 7, one can see that the input to the accumulator is derived from the outputs of the shift register, and thus the binary point of the input may be shifted as will be explained in the next section.

The output of the accumulator is channeled in several directions at different instances of time. At first the information in the accumulator is shifted into the 'A' memory, and then a different word is shifted into the 'B' memory. Then during several intervals of time, the output is fed through the truncation logic to the inputs of certain storage registers. It is not intended at this point to give specifics as to how the information in the accumulator is handled, but one should be aware of the general nature of this data transfer.

C. Shift Register

The shift register serves as a preparatory storage device for the accumulator and also as a means of accomplishing the shifting operation, necessary in the multiplication of the sampled values by their proper coefficients.

The shift register configuration, as shown in Figure 8, is used quite extensively throughout the system and therefore will be discussed in general before proceeding to how it is used in this particular case. Parallel data entry is accomplished through the use of the DC set terminals. A negative pulse as shown in Figure 8 sets that flip-flop to the "one" state which means that there is a logic "one" at the set output terminal, Q, and a logic "zero" at the reset output terminal,

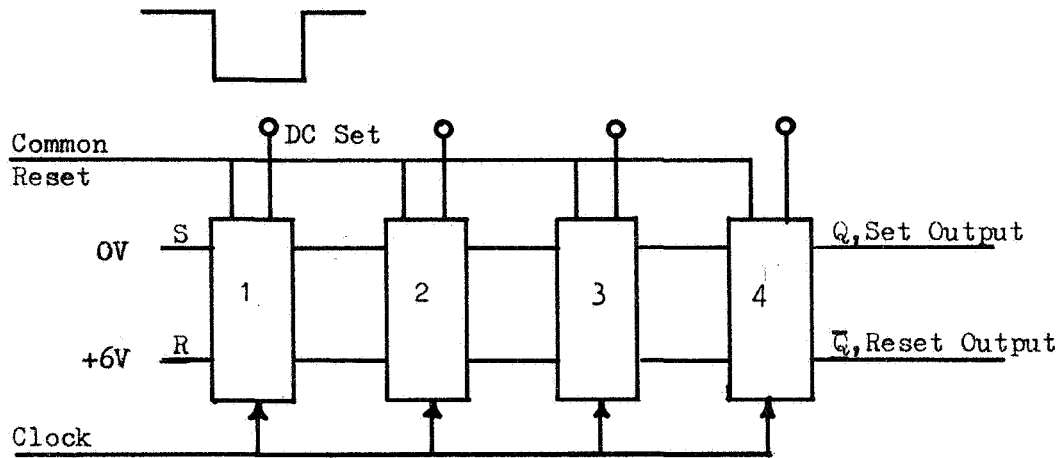


Fig. 8 --Typical Shift Register Configuration

	States of the Flip-flops			
	1	2	3	4
Initial States	0	0	0	0
After 1st FF is Set	1	0	0	0
After 1st Clock Pulse	0	1	0	0
After 2nd Clock Pulse	0	0	1	0
After 3rd Clock Pulse	0	0	0	1
After 4th Clock Pulse	0	0	0	0

Fig. 9 --Transition Table for the Flip-flops

\bar{Q} . When a negative pulse is applied at the common reset terminal, all of the flip-flops are reset to the "zero" state.

Considering only one flip-flop, one can see that when S and R are at logic "one" and "zero" respectively, Q and \bar{Q} go to logic "one" and "zero" respectively with the occurrence of a clock pulse. The reverse is true when the inputs to the flip-flop are reversed. But now consider the case in which the inputs, S and R , to the first flip-flop of the shift register are always at logic "zero" and "one" respectively and information is introduced into the register via the DC set terminals. Each time a clock pulse is applied, the word is shifted one bit to the right. If clock pulses are continuously applied, then the register will automatically be reset to zero since the first flip-flop is constantly feeding in "zeros" to the array. A variation of the above is the case in which the first flip-flop is initially set to the "one" state while the other flip-flops are in the "zero" state. It can be seen from Figure 9 that one clock pulse puts the second flip-flop in the "one" state while all other flip-flops, including the first, are in the "zero" state. With the addition of more clock pulses, a logic "one" can be visualized as moving through the shift register.

Focusing our attention now on the shift register, shown in Figure 10, to be used in this portion of the system one can see that the register will need as many bits to the right of the binary point as the accumulator. Words contained in the storage registers are sequentially selected by the address command processor and routed to the inputs of the shift register. The shifting and adding operation, which constitutes

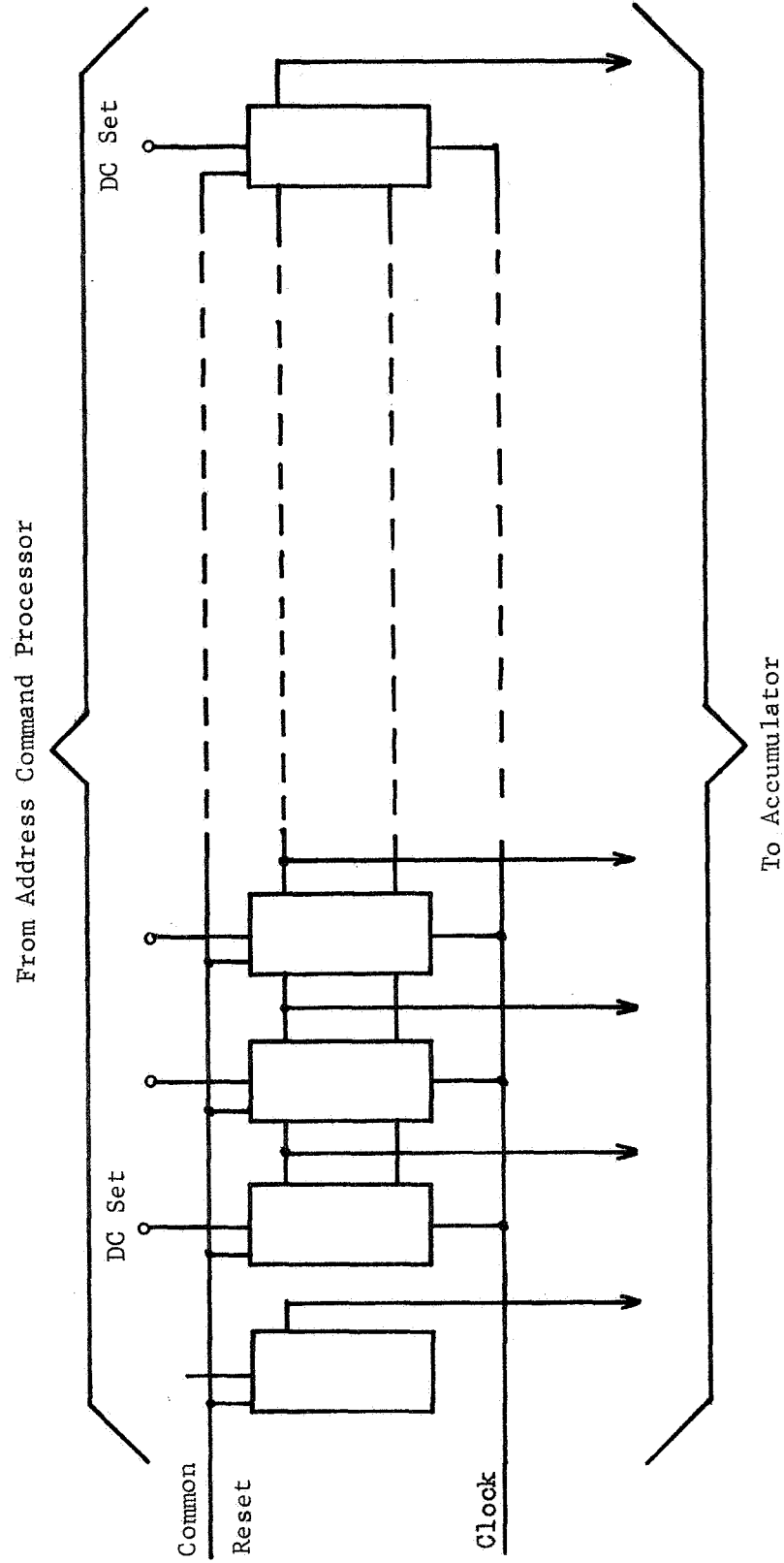


Fig. 10 --Logic Diagram for Shift Register

binary multiplication as discussed in Chapter II, is carried out by the shift register and the accumulator.

D. Storage Registers

The storage registers serve to store the values of the inputs and outputs of the parallel binary accumulator. Figure 11 is an example of one set of registers. The three words which are stored here are the present input, the input T seconds ago, and the input $2T$ seconds ago. The symbol, T , represents the sampling period and therefore a new quantized input is obtained every T seconds. After this occurs, the information in the $e_1(KT)$ register is eventually shifted into the $e_1(KT-T)$ register, and likewise, the $e_1(KT-2T)$ register obtains the information previously stored in the $e_1(KT-T)$ register. In Figure 11 it will be noted that the $e_1(KT-T)$ and $e_1(KT-2T)$ registers have one common clock or data transfer terminal while the $e_1(KT)$ register has a separate terminal. This is because it is desired for $e_1(KT-T)$ and $e_1(KT-2T)$ to take on their proper values before the new input, $e_1(KT)$, is taken into the system.

This concept is employed throughout the system and therefore will be explained in more detail. It was stated in Chapter II that W , X , and Y were to be calculated; then the input was to be sampled, multiplied by a_0 , and added to $W + X + Y$ to yield the present output $e_2''(KT)$. In order to do this, each of the variables required by W , X , and Y must be clocked in advance to realize the correct data transfer.

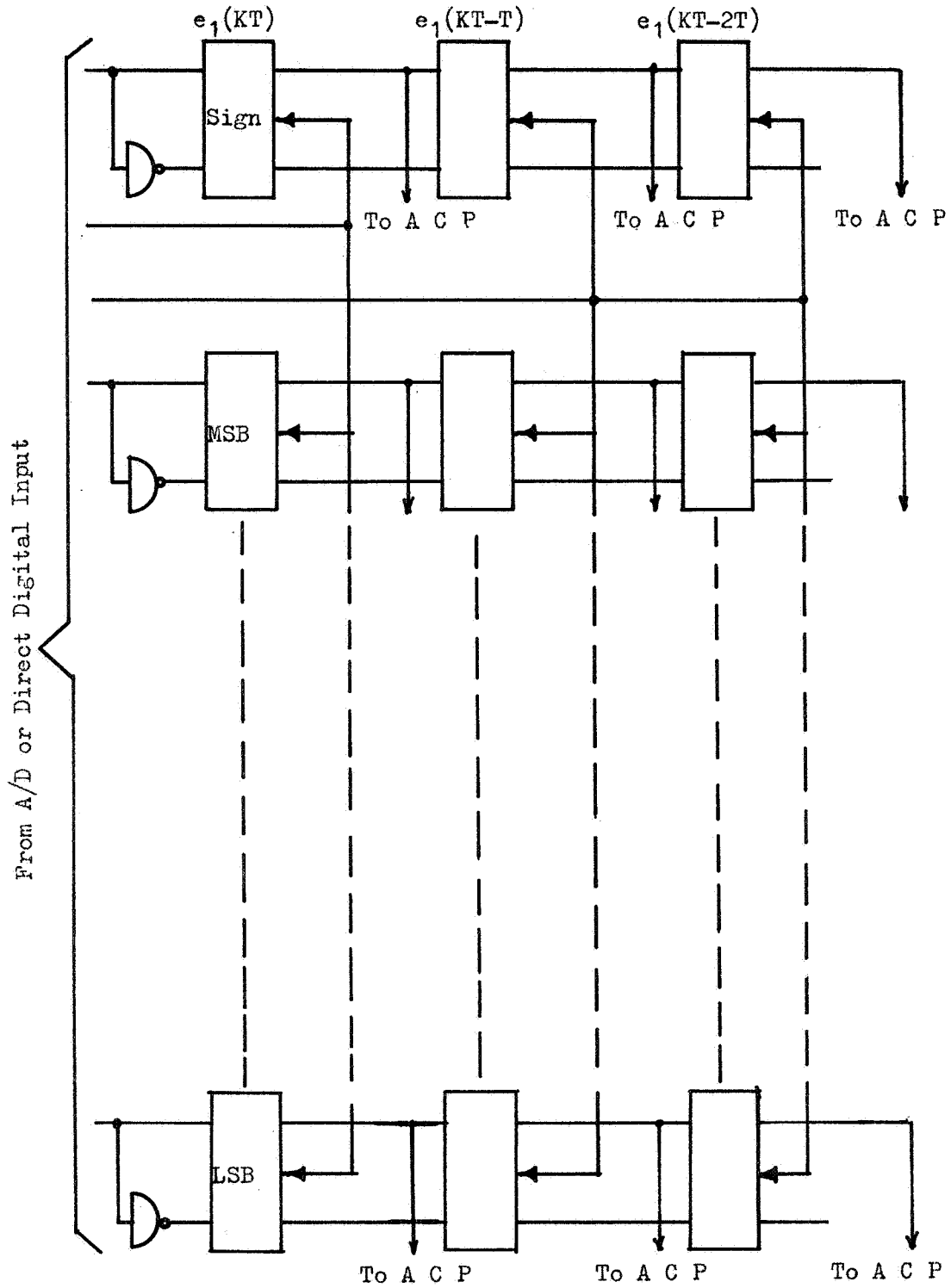


Fig. 11--Typical Storage Register Array

The clocking scheme as shown in Figure 12 illustrates the use of one common clock terminal (B) to accomplish the above. Also it will be noted that several other registers have their own separate clock terminals. A discussion of why this is done will remain for other sections, but as far as data transfer is concerned the system operates basically in the same manner as if one common clock terminal were used for all of the storage registers.

E. Address Command Processor

Referring again to Figure 6, one can see that thirteen storage registers, the 'A' memory, and the 'B' memory must all be sequentially applied to the inputs of the same shift register. This multiplexing or time-sharing problem is solved by the address command processor. Its function is to process an address command from the master controller, to select the proper register, and to channel the contents of the selected register to the DC set terminals of the shift register.

The ACP is shown in Figure 13. Whenever a logic "one" is applied to point 1, the word contained in the selected register, e.g., $e_1(KT)$, appears at the outputs marked D, E, and F. The D output indicates the sign of the selected output which is multiplied by the sign of the coefficient, a_0 . By allowing a "zero" to represent a positive value and a "one" to represent a negative value, one observes that the output of an exclusive-or gate carries the proper sign for multiplication.

The sign output, along with D and E, cannot appear at the output until point 16 receives a positive pulse which loads the data into the

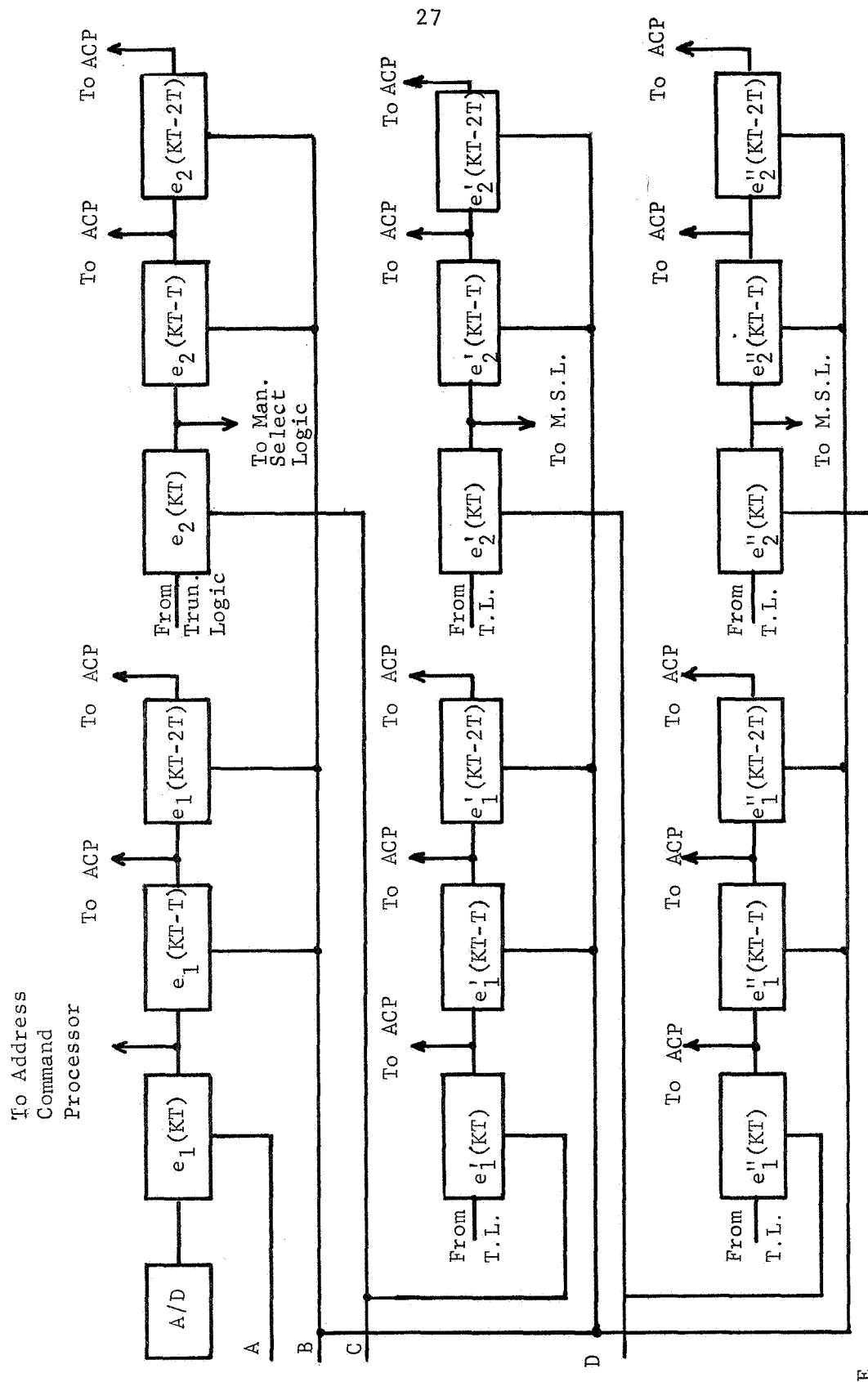


Fig. 12--Clocking Scheme for Storage Registers.

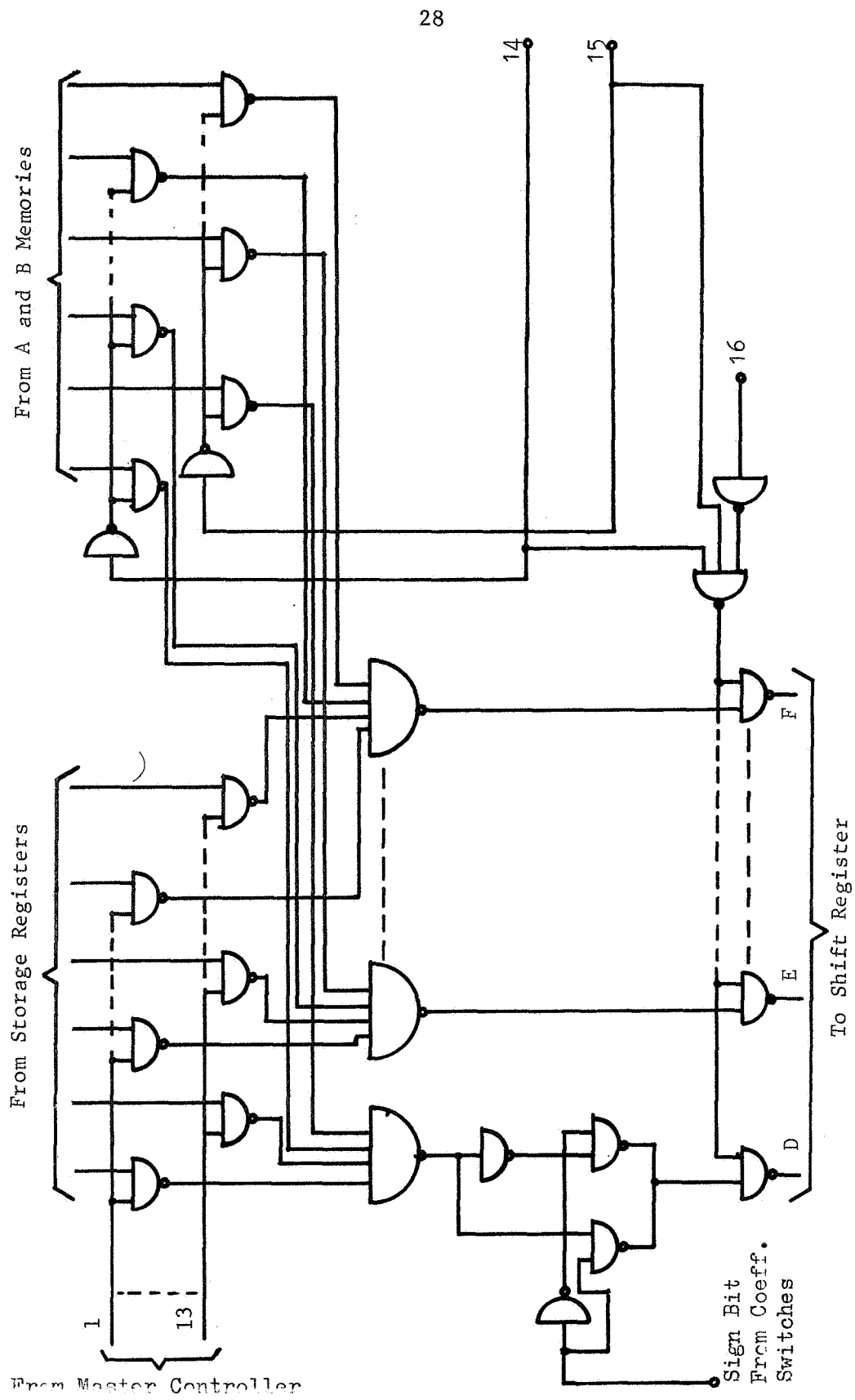


Fig. 13 --Address Command Processor

shift register. A positive pulse implies that the normal level at point 16 is a logic "zero." This in turn clamps the output of the ACP to a logic "one" until a load pulse is received. Therefore no additional information can be introduced into the shift register since the DC terminals of the shift register will be at logic "one."

The ACP also handles the information stored in the 'A' and 'B' memories. The load pulses for the 'A' and 'B' memories are applied at points 14 and 15 respectively, and in this case, these same pulses also serve to address the two memories.

F. Multiplication Logic

Whenever a number [e.g. $e_1(KT-2T)$] is entered into the shift register, it is multiplied by a coefficient [e.g. a_2] by shifting and accumulating this number in the manner discussed in previous sections. The master controller initiates seventeen accumulate pulses and sixteen shift pulses. The purpose of the multiplication logic is to ensure that all of these pulses are processed in the correct manner.

The inputs from the master controller as shown in Figure 14 consist of a series of alternating accumulate and shift pulses. This is to say that input J first receives a pulse, then K receives one, and then the process is repeated for the remainder of the inputs.

The shift inputs are normally at logic "zero," and therefore one can see that a change occurring at any of these inputs will produce a change at the common shift output. The same is true for the accumulate inputs whenever the coefficient switch input for that particular gate is at logic "one."

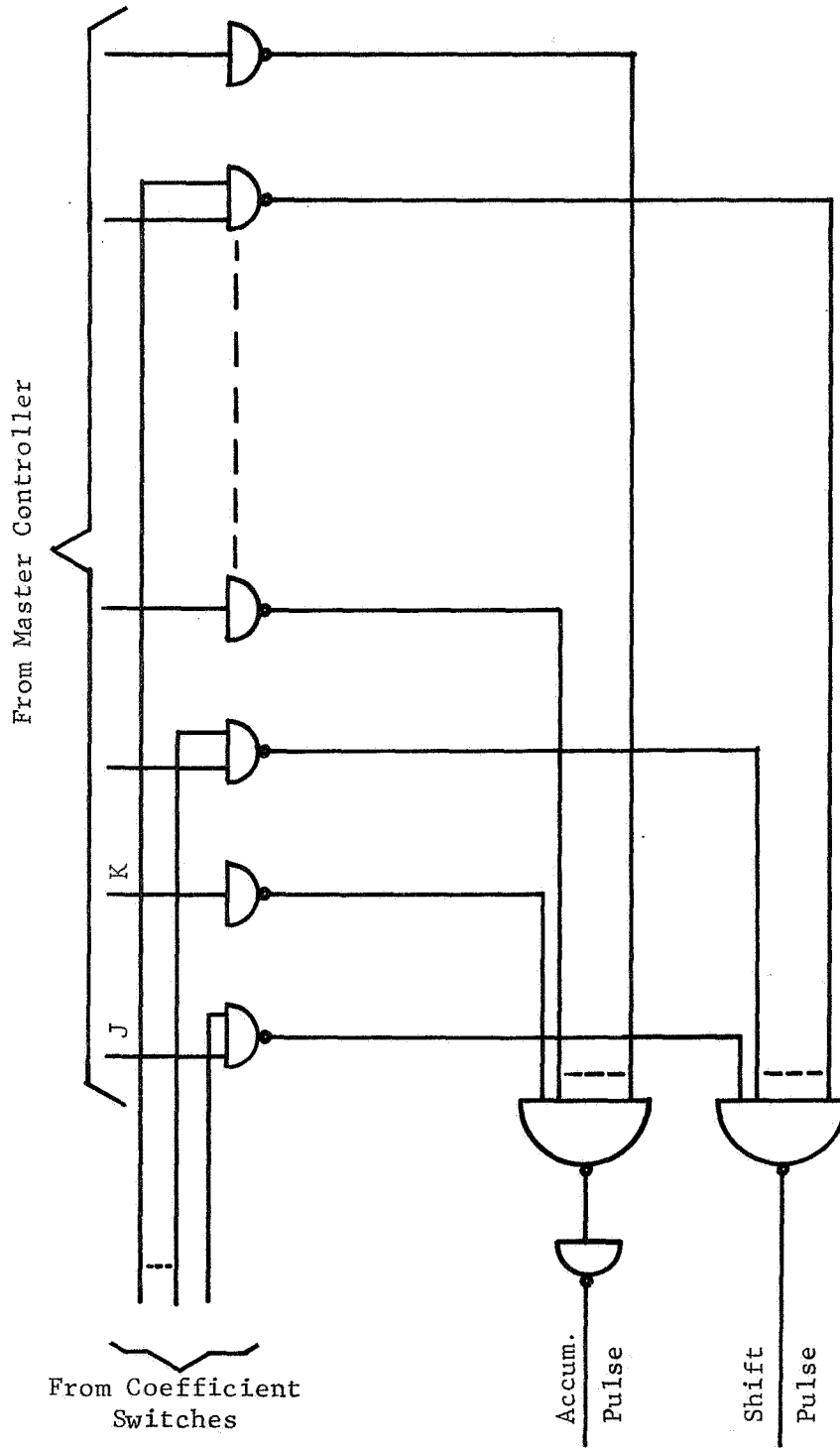


Fig. 14-- Multiplication Logic

If a_2 is given to be 1.0101 for example, it is desired to have an accumulate output only from the first, third, and fifth input accumulate pulses. Therefore, the outputs from the coefficient switches must be at logic "one" to allow these pulses to pass, and the other outputs should be at logic "zero" to disable all other similar gates.

G. Coefficient Switches

In the preceding section, there was only one input from the coefficient switches, and yet thirteen different coefficients must use this same input. The time-sharing logic involved is implemented as shown in Figure 15.

It can be seen that when a given enable input is at logic "one," the levels of that row of coefficient switches appear at the outputs. But if the input is changed to a logic "zero," the positioning of that row of switches will have no effect on the output.

In actual operation only one input at any given instant of time will be at logic "one." In this way, the master controller can activate any given coefficient upon command. The number of bits or switches associated with each coefficient is: a_0 , six bits plus sign; a_1 and a_2 , twelve bits plus sign; and the remainder of the coefficients, seventeen bits plus sign. The above resolution should be sufficient.

H. Truncation Logic

It was stated in a previous section that the accumulator required a total of thirty-six stages with twenty-two of these located to the

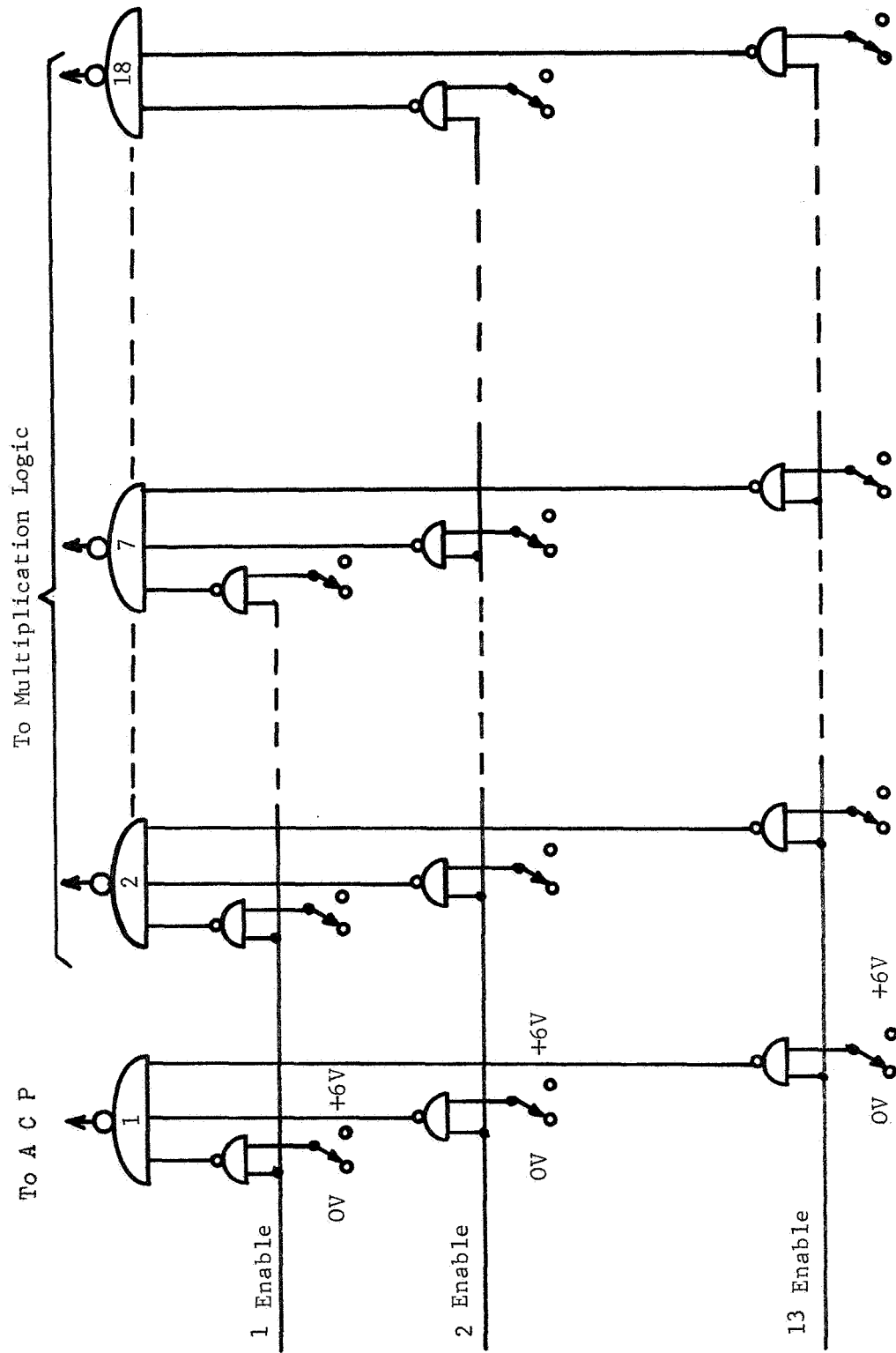


Fig. 15--Coefficient Switches.

right of the binary point. After a series of multiplications and additions, e.g., the computation of $e_2(KT)$, the number in the accumulator is stored in the proper register. It is logical to assume that some of the less significant bits to the far right of the binary point may be omitted before the output is stored, without greatly affecting the system. The ability to do the above is almost a necessity since if an attempt were made to store all of the bits, the accumulator and shift register size would have to increase by about sixteen bits each time a new product was calculated.

Figure 16 illustrates in general how an output is truncated. The number of bits required to the left of the binary point for a given output can be determined for normal operating conditions. But if an extreme input causes a logic "one" to appear to the left of the expected range and logic "zeros" to appear at the normal outputs, then the system acts like an open loop and can go unstable. It is for this reason that combinational logic has been devised as shown in Figure 16 to avoid this situation. Whenever a logic "one" appears outside of the expected range, a logic "one" is applied to all of the regular outputs.

There are six places where quantization or truncation can be realized excluding the input, as shown by the hexagons in Figure 5. The question now is how many bits may be omitted at any given point. The first step in answering this question was to use the $D(z)$ given below as a guideline in designing the system.

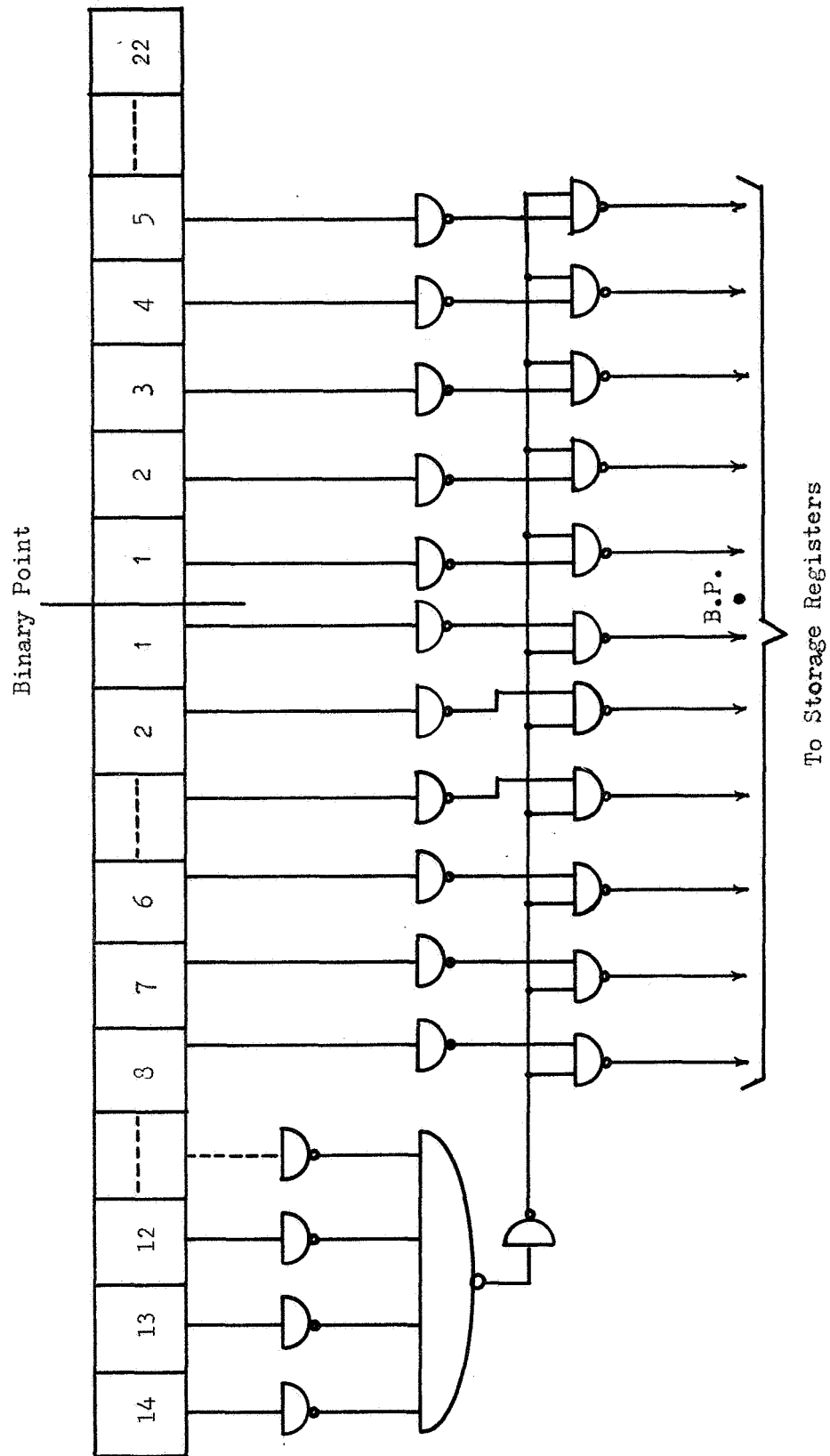


Fig. 16 --Typical Truncation Logic

$$D(z) = \frac{z^2 - 1.7237z + 0.9162}{z^2 - 1.3777z + 0.5316} \times \frac{z^2 - 1.9608z + .9616}{z^2 - 1.9488z + .9512} \times \frac{z - .9960}{z - .9980} \quad (17)$$

A computer program subroutine simulating this compensator was written and inserted into a pre-established master program simulating the flight dynamics of the Saturn V [15]. In effect the system shown in Figure 1 is simulated entirely by one computer program. Then the effects of various quantization techniques on the complete system could be analyzed and then changed until the proper time response was obtained.

As an illustration, $e_2'(KT)$ requires five bits to the right of the binary point and nine magnitude bits to the left. The equation

$$AX = 32 = 2^5$$

listed in Appendix A under Compensator Subroutine implies that there are 5 bits to the right of the binary point, and the equation

$$BX = 16383. / AX = (2^{14} - 1) / AX$$

implies that there are $14 - 5 = 9$ bits left of the binary point. There is a subroutine which performs the quantization that is programmed into the subroutine for the $D(z)$ by insertion of "Call Round" statements.

Figure 5 gives the total number of bits plus the number of bits to the right of the binary point (bits to the right of the binary point is abbreviated as B.R.B.P.) for each point of truncation or quantization within the system.

I. 'A' and 'B' Memories

As stated in Chapter II the value of X is first calculated and stored. The accumulator is then reset and the value of Y is calculated and stored. The storage units for X and Y are the 'A' and 'B' memories respectively, which are both constructed as shown in Figure 17. Data entry in this case is made through the set and reset input terminals with the occurrence of a clock pulse. Whenever a positive pulse occurs at the common reset terminal, all of the flip-flops are reset to the "zero" state.

J. Input Select Logic

Figure 6 might imply that the inputs to the compensator are always in analog form, but since this device is to be a laboratory model, it may be desired to bypass the A/D and have a direct digital input. It is the task of the input select logic to provide the above capability.

Switch B in Figure 18 is a select switch between the digital output from the A/D and the direct digital input. In either case the selected input is fed into the input of the $e_1(KT)$ register.

Before proceeding with actual operation of the entire system, all of the storage registers and flip-flops within the compensator must be reset. Ordinarily this would involve resetting each flip-flop with the aid of the DC reset terminal, but this method is awkward because of the great number of memories involved. Instead, Switch A in Figure 18 is set to logic "zero" which in turn clamps all of the inputs to the $e_1(KT)$ register to logic "zero." The sampling clock is then turned on for a

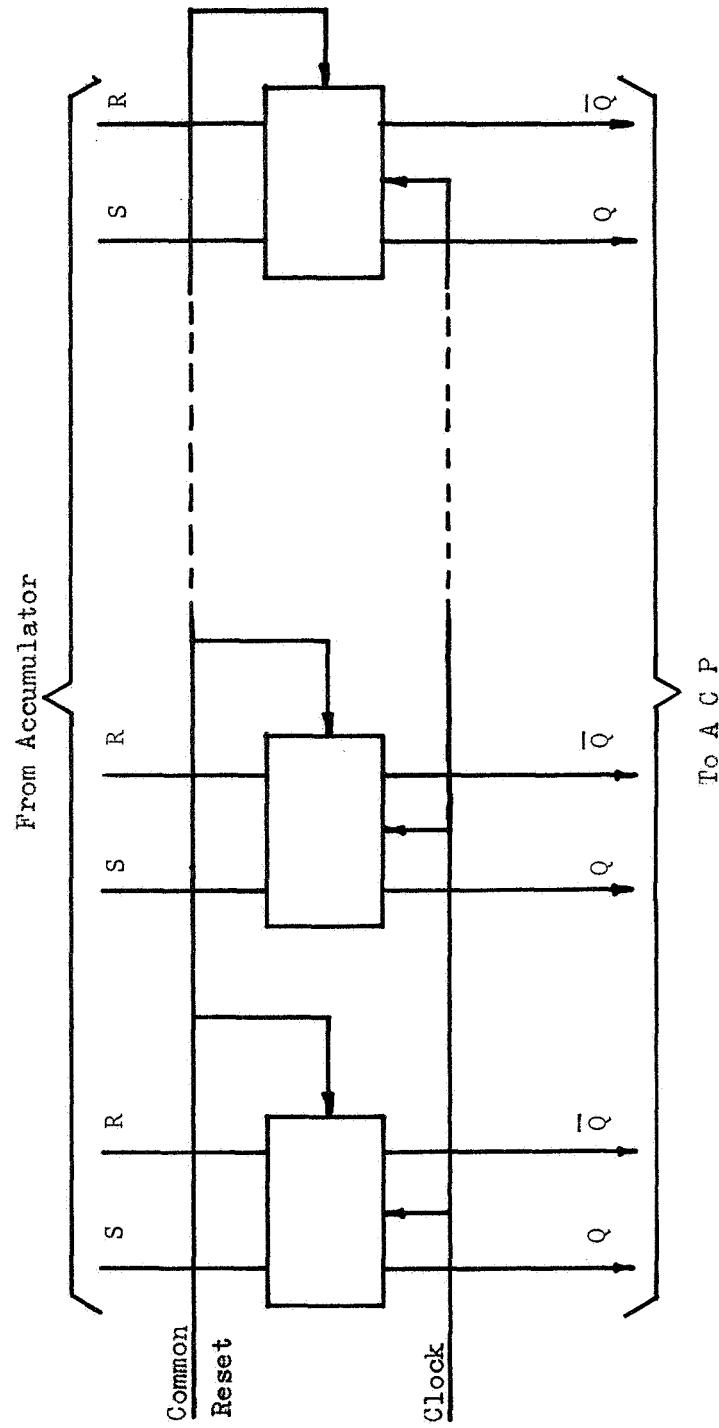


Fig. 17 --Logic Diagram for Both A and B Memories

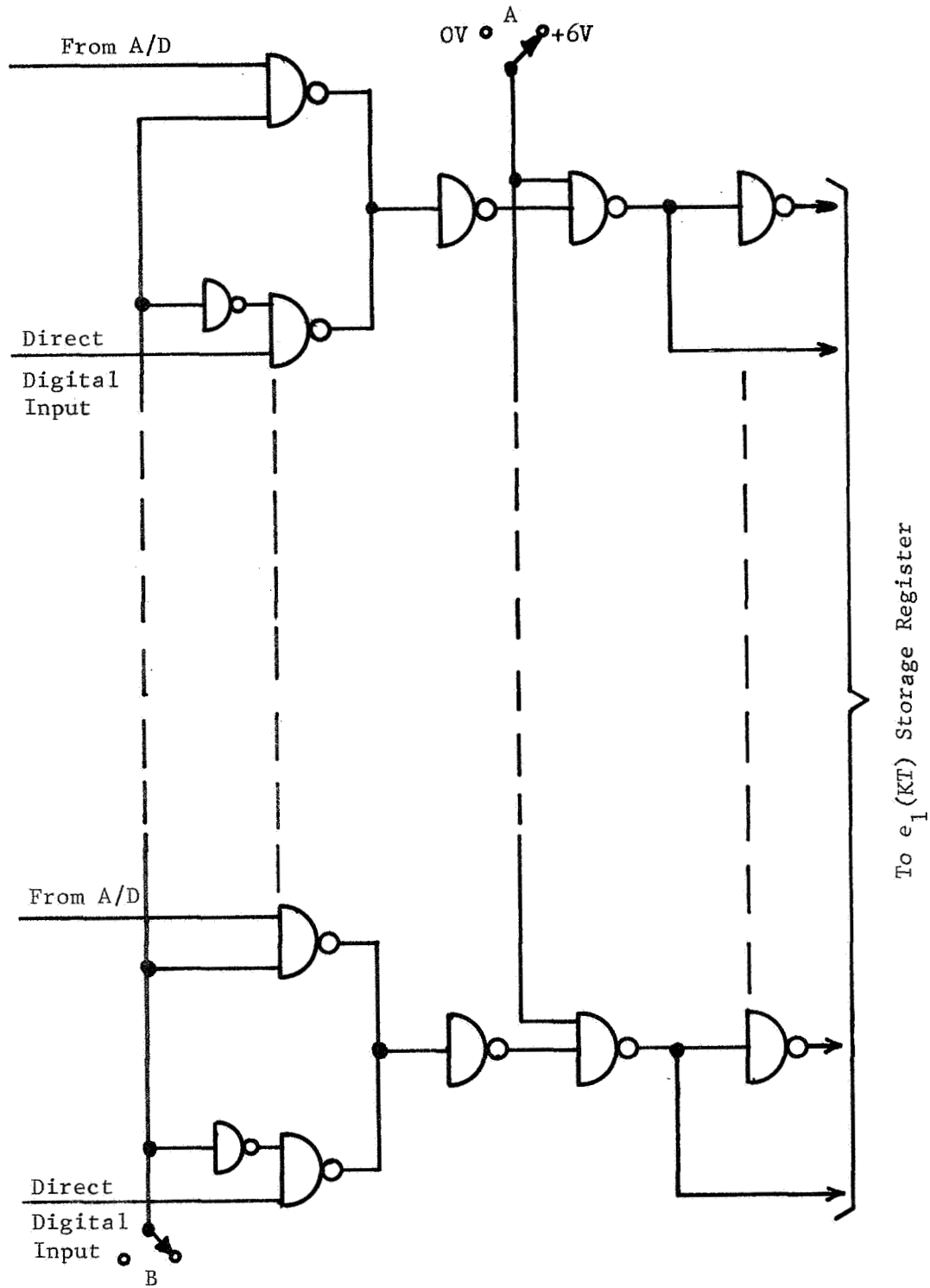


Fig. 18 --Input Select Logic

few seconds after which all of the flip-flops are reset since a series of zero inputs was processed by the system with $D(z) = + 1.0$.

K. Manual Select Logic

For any given input, the outputs, $e_2(KT)$, $e_2'(KT)$, and $e_2''(KT)$, are presented to the input of the manual select logic within a few microseconds of each other. The manual select logic is controlled by three switches as shown in Figure 19. The switch corresponding to the desired output is set to logic "one," and the other two switches are set to logic "zero." In this way, only the desired signal reaches the output.

Although the compensator will not be built this way, it can be seen from the above discussion and Figure 6 that three outputs from a second, fourth, and sixth order compensator could be obtained almost simultaneously. This capability could be of some value in future applications.

L. Master Controller

Up until this point each logic network has been explained separately in terms of what function is performed when an input is either a series of pulses or a logic level, but no mention has been made as to the origin of these signals. It is the purpose of the master controller to issue these commands and to ensure that information is being processed in the correct sequence and manner throughout the system. Due to its involved nature, the master controller, shown in Figure 20, will be explained as a unit unto itself before detailing how it operates on the

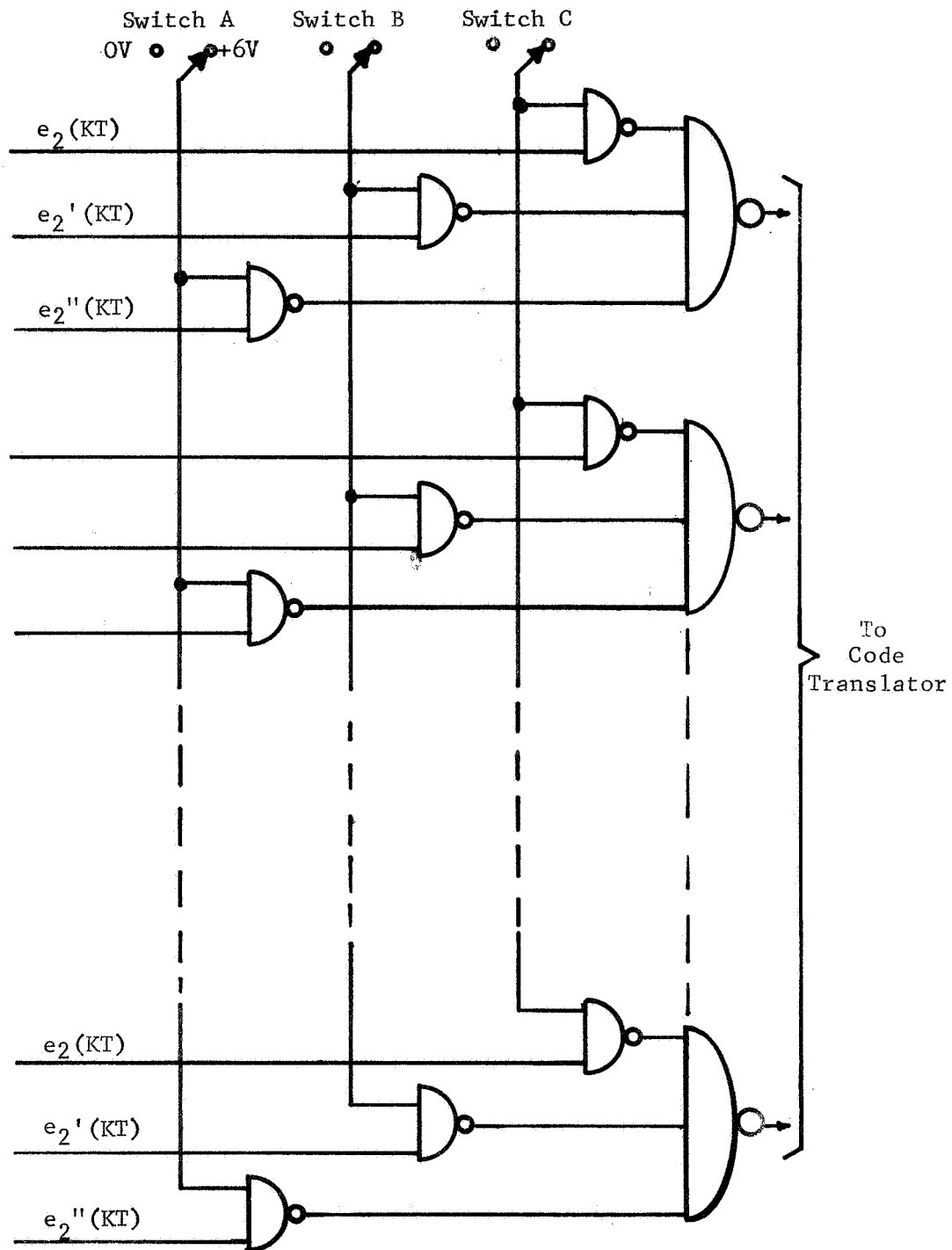


Fig.19--Manual Select Logic

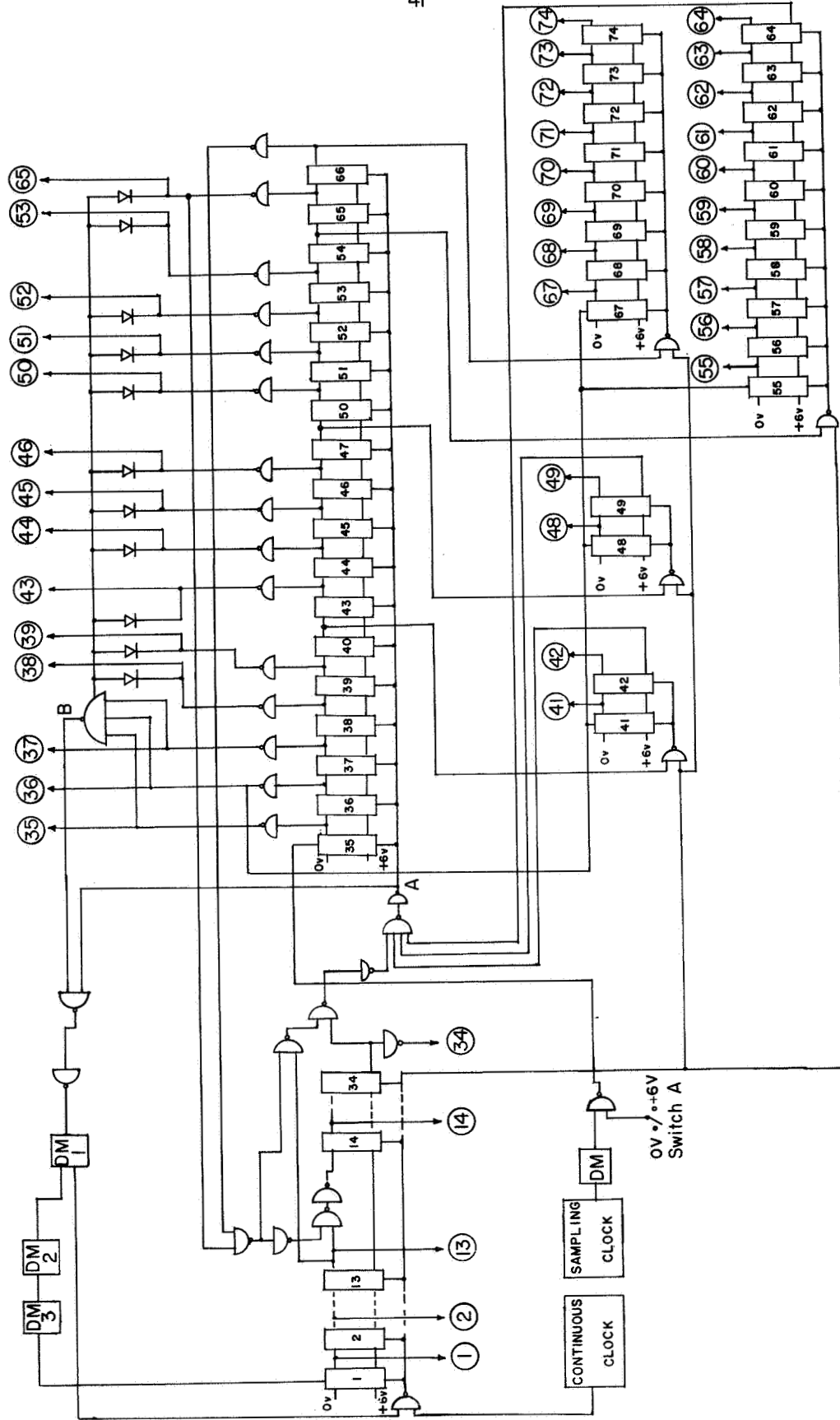


Fig. 20--Logic Diagram of Master Controller

other networks. Each of the sets of shift registers uses the previously mentioned technique of shifting a logic "one" through the register. In order to facilitate the explanation of the logic functions, each of the flip-flops and outputs are numbered as nearly as possible in the order in which they go to the one state, or have a logic "one" at its set output.

It can be seen that point B is the output of a fourteen-input NAND gate. Its inputs are derived from the set outputs of flip-flops thirty-five through thirty-nine, forty-three through forty-six, fifty through fifty-three, and sixty-five. When the above flip-flops are all in the "zero" state, the output of the big NAND gate is a logic "zero." But if any one of those flip-flops is in the "one" state, then the output is a logic "one." This output is used as an enable or disable signal for the output from point A. In other words the pulses from point A will reach delay-multivibrator number one only when the output of the fourteen-input NAND gate is at a logic "one."

Since the continuous clock is always applied to the controller, some means must be found to disable this clock, set flip-flop one to the "one" state, and then enable the clock again. If flip-flop one were reset without stopping the clock, then there is a good possibility that several of the first few flip-flops would be set to the "one" state when only one such flip-flop is desired. Now when the input to delay-multivibrator "one" experiences a logic "zero" to "one" transition, the three delay elements produce output pulses as shown in Figure 21. The assertion output of number one triggers the input to

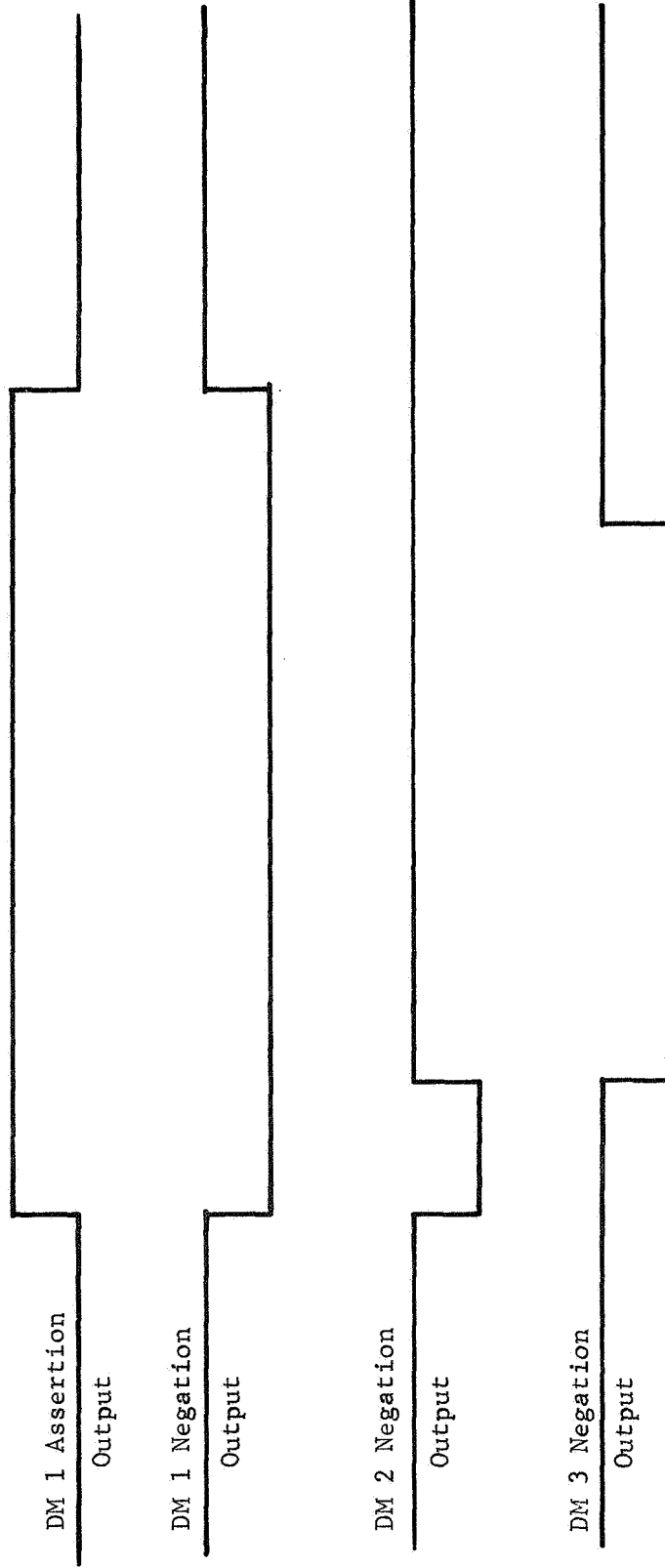


Fig.21--Timing Pulses for Master Controller Delay-Multivibrators.

number two while the negation output is simultaneously disabling the continuous clock. The output of number two triggers the input of number three, whose negation output in turn sets flip-flop one in the "one" state. After this has occurred, the wide negation pulse of number one returns to logic "one" thereby enabling the continuous clock again. Number two is inserted to ensure that the clock is disabled before the flip-flop is set. This "one" state is clocked down the line through flip-flop thirty-four. For the moment the logic between flip-flops thirteen and fourteen can be ignored. Now when the logic "one" passes through flip-flop thirty-four, a pulse appears at point A since the other inputs to the four-input NAND gate are at logic "one." If flip flop thirty-five was previously in the "one" state, the pulse at point A would send it to the "zero" state and simultaneously send flip-flop thirty-six from the "zero" to the "one" state. This same pulse would also activate the three delay-multivibrators which would reset flip-flop one to the "one" state and repeat the cycle.

The cycle is temporarily discontinued whenever flip-flop forty goes to the "one" state because now all of the inputs to the fourteen-input gate are at logic "one," its output is at logic "zero," and therefore the gate controlling the input to delay-multivibrator one is disabled. However, when flip-flop forty goes to the "one" state, this means that the set output goes to logic "one." When the above occurs, the wire connected to the set output permits the gate controlling the continuous clock input for flip-flops forty-one and forty-two to be enabled. Flip-flop forty-one, along with flip-flops forty-eight, fifty-

five, and sixty-seven, had been set to the one state when the logic "one" had passed through flip-flop thirty-six. So the logic "one" is shifted out of flip-flop forty-one and through flip-flop forty-two whenever the continuous clock gate is enabled. Now when the reset output of flip-flop forty-two undergoes a one-zero-one type transition, a pulse is once again generated at point A which shifts the logic "one" out of flip-flop forty and into forty-three. In the process, flip-flop one is also set to the "one" state and the cycle is repeated until flip-flop forty-seven is reached. At this point and also at flip-flop fifty-four, the same type procedure is carried out as previously described.

At this point, the master controller will now be examined in relation to actual system functions. The sampling clock runs at a relatively slow frequency. Its output is fed into a delay-multivibrator whose output in turn consists of a series of short pulses with the same period as the input. Switch A is provided to inhibit the flow of pulses whenever desired. The purpose of the sampling clock is to set flip-flop thirty-five to the one state at such intervals as to allow the complete cycle to occur before resetting the flip-flop to the "one" state again. If all of the flip-flops are initially in the "zero" state and a sampling clock pulse is applied, then output thirty-five experiences a logic "one-to-zero" transition, and output B undergoes a reverse transition which triggers the delay-multivibrators. This is the only case in which the output of the fourteen-input NAND gate is used as a triggering signal. Output thirty-five is connected to terminal B in Figure 12. This terminal provides the initial shift required for the

storage registers since the input, $e_1(KT)$, is obtained at a later time. The logic "one" is shifted also through the first thirty-four flip-flops with no other effect than transferring flip-flop thirty-six from the "zero" to the "one" state. Now, output thirty-six addresses the $e_1'(KT-T)$ storage register and prepares $e_1'(KT-T)$ for entry into the storage register. This same output also addresses the proper terminal in Figure 15 which allows the coefficient a_1' to operate on the multiplication logic. After both the proper coefficient and sampled value to be multiplied together have been selected, flip-flop one loads the contents of the $e_1'(KT-T)$ storage register into the shift register. Then flip-flops two through thirty-four provide alternating accumulate and shift pulses during which the number in the shift register is actually accumulated only when the input gates of Figure 14 have been enabled by the switches corresponding to the coefficient, a_1' . In effect, flip-flops two through thirty-four have controlled the multiplication of a_1' by $e_1'(KT-T)$.

Outputs thirty-seven, thirty-eight, and thirty-nine perform the same type operation on $e_1'(KT-2T)$, $e_2'(KT-T)$, and $e_2'(KT-2T)$ and their corresponding coefficients. The sum of these products is stored in the accumulator, which represents the value of X. Output forty-one shifts X out of the accumulator and stores it in the 'A' memory. Output forty-two resets the accumulator.

Outputs forty-three through forty-six and fifty through fifty-three perform the computations of Y and W respectively in a similar manner. However, when the value of W is calculated, it is not stored as X and

Y were but remains in the accumulator for the time being. Outputs fifty-five through sixty-two furnish the eight pulses required for the analog-to-digital conversion process. Output sixty-three shifts the value of $e_1(KT)$ out of the A/D and into the $e_1(KT)$ storage register. The A/D is then reset and prepared for the next sampling period by output sixty-four.

Now that a new value of $e_1(KT)$ has been obtained by the system, output sixty-five addresses the $e_1(KT)$ storage register, and it also addresses the set of coefficient switches controlling the value of a_0 . Output one then loads the contents of the $e_1(KT)$ register into the shift register, and since a_0 is to have six bits plus sign, only outputs two through thirteen operate on the multiplication logic.

As soon as the product of a_0 and $e_1(KT)$ is calculated, it is desired to have an output, $e_2''(KT)$, almost immediately. This is the reason for the logic installed between flip-flops thirteen and fourteen which allows the logic "one," being shifted through the registers, to bypass flip-flops fourteen through thirty-four and to appear directly as an output at point A. If this were not done, a certain amount of unnecessary propagation time would be introduced into the system. When the last pulse appears at point A, it causes flip-flop sixty-six to enable the continuous clock gate for flip-flops sixty-seven through seventy-four.

Output sixty-seven shifts the value of $e_2(KT)$ out of the accumulator, through the reduction logic, and into the $e_1'(KT)$ storage register. This output is also made available to the input of the manual select

logic. The shift register is also cleared by output sixty-seven.

The value of X which is stored in memory 'A' is entered directly into the shift register by output sixty-eight through the use of terminal 14 in Figure 13. This same output also resets some of the less significant stages of the accumulator to the "zero" state; this is tantamount to truncating the value of $e_1'(KT)$, the input to the second stage. The value of X is then accumulated by a pulse from output sixty-nine. The shift register is then cleared by output seventy using the common reset terminal in Figure 10. Also this same output truncates $e_2'(KT)$. The accumulator contains the value of $e_2'(KT)$, which is shifted into the $e_1''(KT)$ register by output seventy-one. It is also shifted through the truncation logic to the input of the manual select. In addition, output seventy-one resets certain stages of the accumulator and thereby truncates the value of $e_1''(KT)$, the input to the third stage.

The value of Y which is stored in the 'B' memory is entered into the shift register by output seventy-two being applied to terminal 15 in Figure 13. Output seventy-three truncates $e_2''(KT)$ and also clears the shift register. Output seventy-four shifts $e_2''(KT)$ through the truncation logic to the input of the manual select logic and also the accumulator is presently reset to prepare the system for the next sampling period.

IV. THEORETICAL RESULTS

A. System Response

The purpose of the $D(z)$, as stated in the introduction, is to provide compensation for the hybrid simulation of the Saturn V thrust vector control system. The extent to which it accomplishes this is determined by simulating the system with a computer program and then creating certain disturbances so that the system response in returning to the stable state could be observed. A program for the flight dynamics was (see Appendix A) used in conjunction with a program for the $D(z)$, to simulate the complete system [15].

Figure 22 gives a plot of the error signal, ϕ_d , in degrees versus time. This response was obtained using the fifth-order compensation function given below and an initial condition on ϕ of ten degrees [15].

$$D(z) = \frac{z^2 - 1.7237z + 0.9162}{z^2 - 1.3777z + 0.5316} \times \frac{z^2 - 1.9608z + 0.9616}{z^2 - 1.9488z + 0.9512} \times \quad (18)$$

$$\frac{z - 0.9960}{z - 0.9980}$$

It can be observed that the settling time is 20 sec. Near the end of the flight simulation, ϕ_d is below one tenth of a degree.

Figure 23 gives the time response of the system using the sixth-order function given below:

$$D(z) = \frac{z^2 - 1.88z + 0.948}{z^2 - 1.559z + 0.60598} \times \frac{z^2 - 1.517z + 0.888}{z^2 - 1.453z + 0.51906} \times \frac{z^2 - 1.988z + 0.9884288}{z^2 - 1.50256z + 0.5032742} \cdot \quad (19)$$

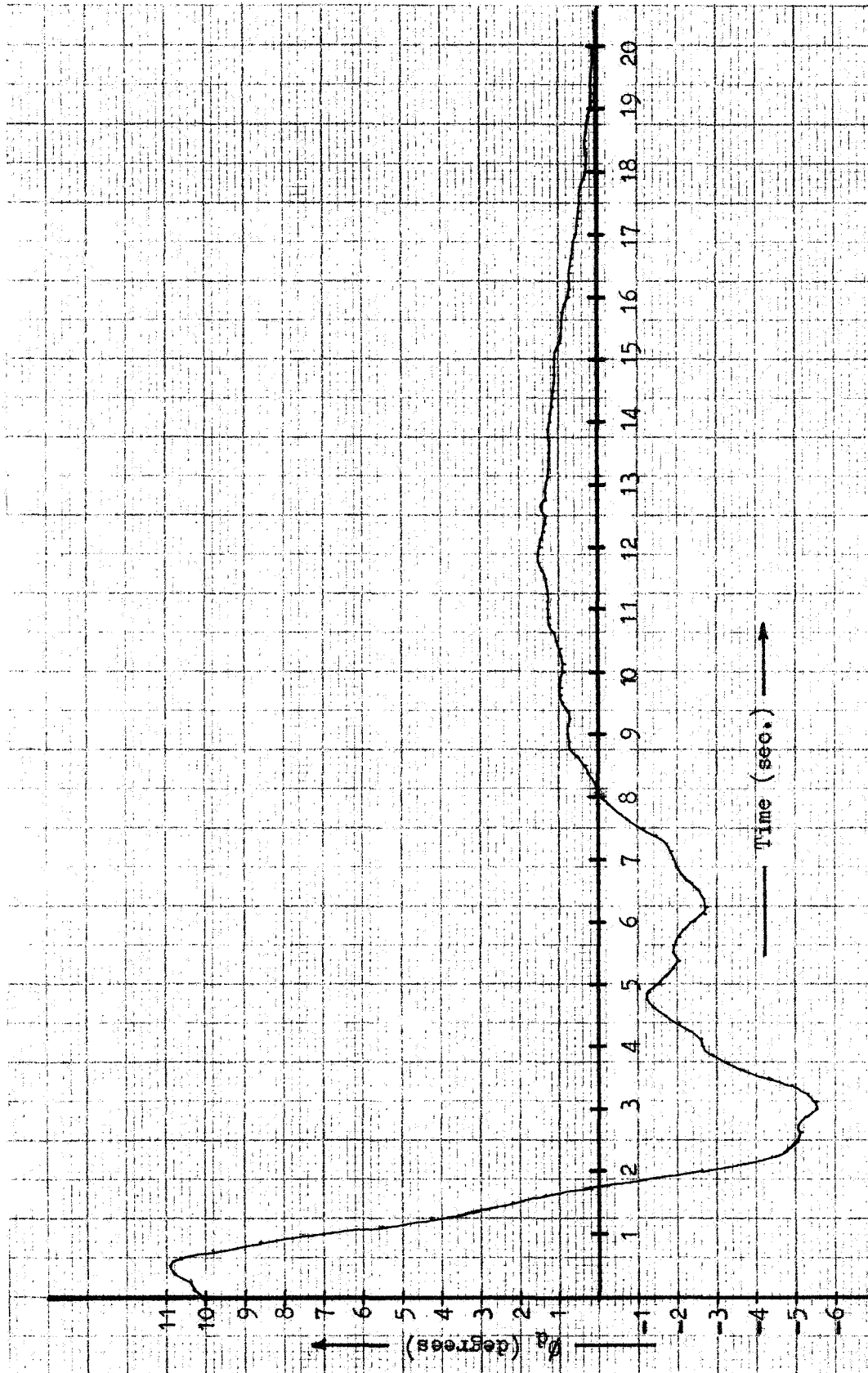


Fig. 22 --Time Response for Fifth-Order Compensation Function

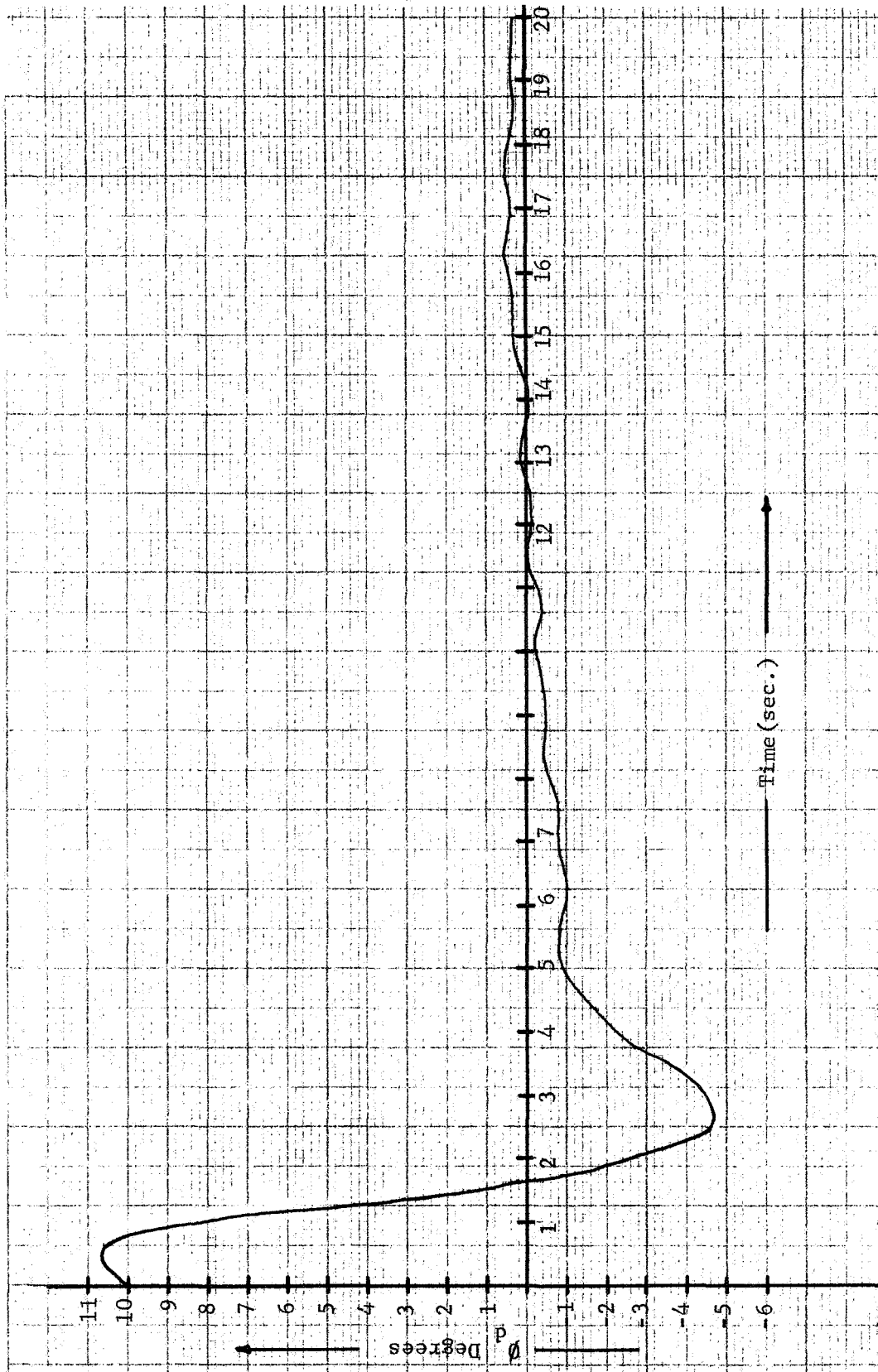


Fig. 23--Time Response for Sixth-Order Compensation Function

The gain for this case is 1.69 instead of 1.0 as before. Also the feedback term $e_2'(KT)$ is reduced from sixteen bits plus sign with five bits to the right of the binary point to eleven bits plus sign with zero bits to the right of the binary point. It was found that the response was improved when these five bits to the right of the binary point were not used.

The final value of ϕ_d was approximately one half of a degree, which is not as good as the first case due at least in part to the fact that the sixth-order function has inherent properties which make it difficult to realize under any conditions. Also, the $D(z)$ was designed using the fifth-order function as a guideline to evaluate system performance.

B. Sampling Frequency and Delay Considerations

The maximum sampling frequency requirement is 25 Hz, but the machine is capable of operating much faster. Figure 4 illustrates that the minimum sampling period is equal to $T_c + T_x + T_y + T_w$.

The maximum T_c is the time required for the logic "one" to travel from flip-flop fifty-five to flip-flop seventy-four in Figure 20. This propagation time is determined by the frequency of the continuous clock which has a period of ten microseconds. This value is chosen to allow the accumulator sufficient time to process its carry signals. The value of T_c is found to be equal to 0.46 milliseconds. T_c represents the time delay between input and output, and is small compared with a normal sampling period of 40 milliseconds.

The time required for the logic one to reach flip-flop fifty-four in Figure 20 is equal to $T_x + T_y + T_w$, which is 4.95 milliseconds. Therefore the minimum sampling period is approximately 5.5 milliseconds, or the maximum sampling frequency is 182 Hz.

V. USE OF THE COMPENSATOR AS A PIECE OF LABORATORY EQUIPMENT

This chapter is intended to give a person not familiar with digital equipment a step-by-step procedure for setting up any desired compensation function. The $D(z)$ to be realized should be in the form given below, where A_0 is used to vary the DC gain.

$$D(z) = \frac{A_0 z^2 - A_1 z + A_2}{z^2 - B_1 z + B_2} \times \frac{z^2 - A_1 P z + A_2 P}{z^2 - B_1 P z + B_2 P} \times \frac{z^2 - A_1 P P z + A_2 P P}{z^2 - B_1 P P z + B_2 P P} \quad (20)$$

These coefficients are then fed into a computer program given in Appendix B; this program converts them to binary numbers. These binary numbers then denote how each of the coefficient switches in Figure 24 are to be set. A_0 has an approximate range of ± 2 which should be sufficient. If A_0 is different from ± 1 , then the corresponding values of A_1 and A_2 must be multiplied by A_0 . For instance, if A_0 were 1.5, then the value of A_1 and A_2 entered into the computer program would be $A_0 \cdot A_1^*$ and $A_0 \cdot A_2^*$, where the asterisk indicates the value of A_1 and A_2 as they appear in Eq. (20). The settings for the fifth-order compensator are given in Appendix B.

The type of input desired is selected using the indicated switch, on the control panel as shown in Figure 24. The order of the compensator to be realized is determined by the three switches marked manual select logic. The switch corresponding to the desired output must be

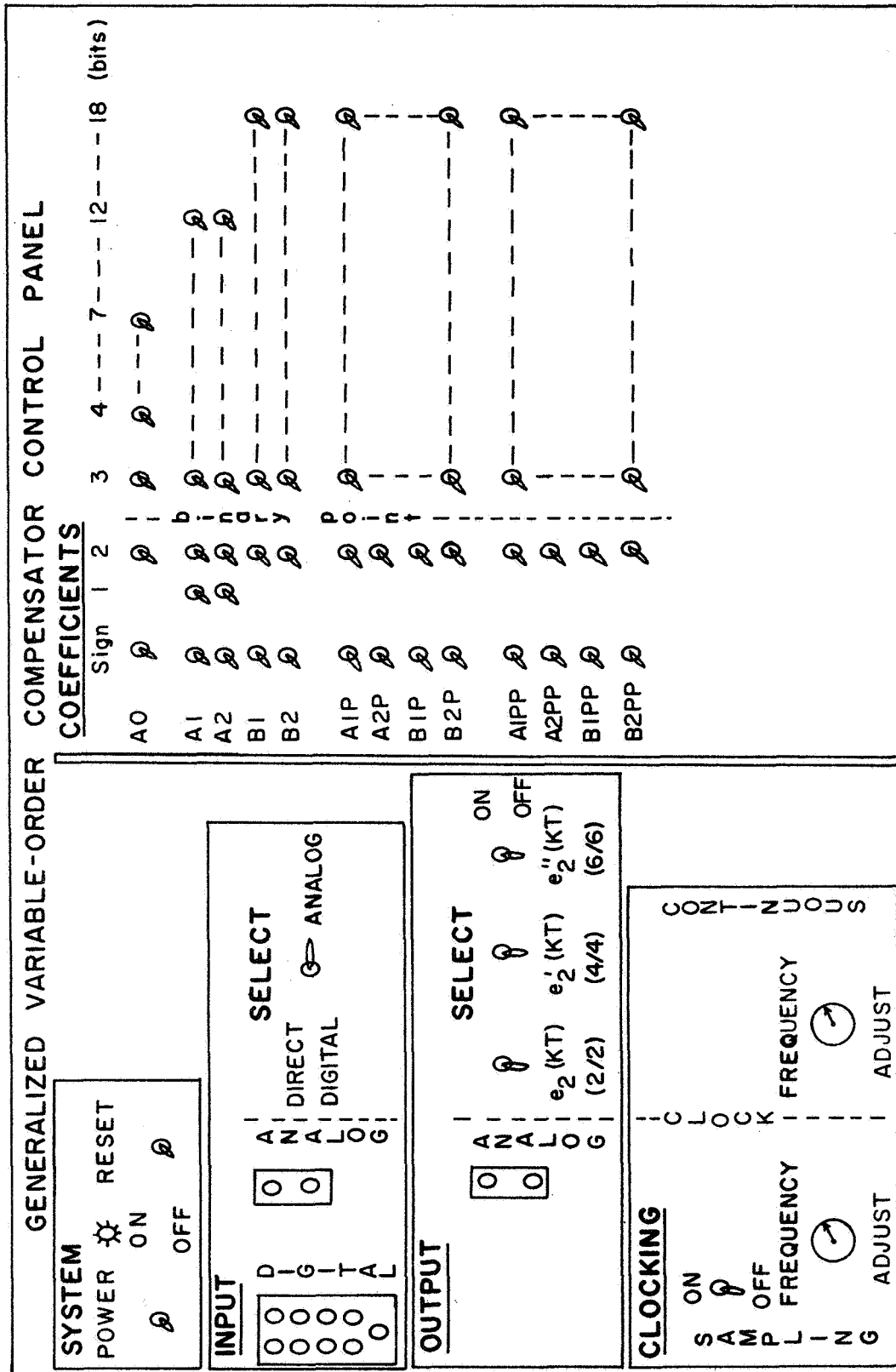


Fig. 24--Control Panel Layout

in the ON position, and the other two must be in the OFF position.

As a last consideration, it should be noted that the sampling and continuous clock frequency can be varied.

The sequence of events to be followed in setting up a compensation function is as follows:

- 1) Connect the desired input and position the input select switch accordingly.
- 2) Connect the analog output.
- 3) Using the output select switch, pick the desired output.
- 4) Adjust the continuous clock and sampling frequency controls.
- 5) Set the power switch to the ON position.
- 6) Set the coefficient switches for $D(z) = + 1.0$.
- 7) Set the system reset switch to the ON position.
- 8) Set the sampling clock switch to the ON position for five seconds and then set it to the OFF position.
- 9) Set the system reset switch to the OFF position.
- 10) Using the computer program, set the coefficient switches to the correct positions.
- 11) Set the sampling clock switch to the ON position and system operation begins.

If the system operation is not satisfactory, it may be necessary to rearrange the positioning of some of the terms in Eq. 20. This manipulation sometimes proves advantageous as a result of the method by which the internal truncation was realized.

VI. CONCLUSIONS

The machine as presented here should serve as a useful tool for one involved in the task of determining which compensation function is optimum for a given system. In this respect the device has the possible limitation that the desired optimum compensation functions may not be realizable with the machine. This results from the necessary utilization of one function as the main guideline in achieving an acceptable design, although other functions were considered to provide maximum capability.

Another factor which contributed to the above problem was truncation or round-off. It would appear logical for the compensator to be more accurate as the amount of truncation was decreased, but this was not always the case, as was cited in Chapter IV. In one example, a much better response was obtained when the number of output bits from the second stage was reduced from 17 to 12.

The laboratory-oriented special purpose computer proposed in this paper is versatile in that it can easily be adapted to serve in many diverse sampled-data systems.

REFERENCES

1. C. C. Carroll, H. H. Hull, D. L. Chenoweth, H. T. Nagle, Jr., and W. L. Oliver, Jr., "On the realization of a generalized second-order digital compensator," Auburn Research Foundation, Auburn University, Auburn, Alabama. Technical Report No. 9, Contract No. NAS8-11274, September, 1967.
2. C. C. Carroll, J. W. Jones, Jr. H. T. Nagle, Jr., G. E. Jordan and W. L. Oliver, Jr., "The hybrid realization of a digital controller for the PIGA control loop," Auburn Research Foundation, Auburn University, Auburn, Alabama. Technical Report No. 6, Contract No. NAS8-20163, September, 1967.
3. C. L. Phillips, R. K. Cavin, III, D. L. Chenoweth, and J. C. Johnson, "Digital compensation of the thrust vector control system," Auburn Research Foundation, Auburn University, Auburn, Alabama. Technical Report No. 7, Contract No. NAS8-11274, January, 1967.
4. H. T. Nagle, "The organization of a special-purpose computer to implement a generalized digital filter for sampled-data systems," Doctoral Dissertation, Auburn University, Auburn, Alabama, June, 3, 1968.
5. J. N. Salzer, "Frequency analysis of digital computers operating in real time," Proceedings of the IRE, Vol. 42, February, 1954 pp. 457-466.
6. M. A. Martin, "Frequency domain applications to data processing," IRE Transactions on Space Electronics and Telemetry, Vol. SET-5, no. 1, March 1959, pp. 33-41.
7. F. Kuo and J. F. Kaiser, System Analysis by Digital Computer, New York, N. Y.: John Wiley and Sons, Inc., 1966.
8. L. B. Jackson, J. K. Kaiser, and H. S. McDonald, "Implementation of digital filters," IEEE International Convention-68, New York, N. Y., March, 1968.
9. A. V. Oppenheim and T. G. Stockham, "Application of homomorphic filtering," IEEE International Convention-68, New York, N.Y., March, 1968.

10. R. W. Kearns, "A digital compensator for automatic control systems," Grant NsG-36-60, Case Institute of Technology, Cleveland, Ohio, June, 1964.
11. D. J. Gawlowicz, "A programmable digital compensator for single-loop high performance digital servomechanisms," Grant NsG-36-60, Case Institute of Technology, Cleveland, Ohio, 1965.
12. Y. Cho and F. G. Popp, "A high-speed arithmetic unit for digital filter implementation," NEREM-67, Boston, Mass., November, 1967.
13. C. C. Carroll and R. White, "Discrete compensation of control systems with integrated circuits," IEEE Transactions on Automatic Control, vol. AC-12, vol. 5, October, 1967, pp. 79-82.
14. H. T. Nagle and C. C. Carroll, "Organizing a special-purpose computer to realize digital filters for sampled-data systems," (to be published).
15. C. L. Phillips, et. al., "Digital compensation of the thrust vector control system," Technical Report Number 10, NAS8-11274, Auburn Research Foundation, Auburn, Ala., January, 1968.

APPENDICES

The Fortran Computer Programs utilized in this study are included in the appendices for reference.

APPENDIX A

FORTRAN SOURCE PROGRAM FOR SIMULATING THE FLIGHT DYNAMICS AND COMPENSATOR

SOURCE DECK

```

C      THE MASTER PROGRAM SIMULATES THE FLIGHT DYNAMICS.
      DIMENSION TX1(8),TY1(8),TY2(8),X1(2002),Y1(2002),
      1Y2(2002)
      DATA TY2/48HE1(1)
      1/
      DATA TY1/48HPHID
      1/
      DATA TX1/48HTIME
      1/
      NN=100
      DIMENSION WB(4),ZETA(4),GM(4),YB(4),YPB(4),YPD(4),
      1 A(14,14),AN(14,
      114,25),B(14,14),C(14,14),AI(14,14),AT(14,14),AIN(14,
      1 14,25),AANT(
      2 14,14),BD(14),X(14),X2(14),CONTRL(2001)
C
C      D(Z) UNCOUPLED FROM DISCRETIZED SYSTEM TRANSITION
C      MATRIX
C
C      QUANTIZATION EXCLUDED
C
      COMMON/COM1/E1(3),E2(3),E1P(3),E2P(3),E1PP(3),E2PP(3)
      CALL TRAP
C
C      STORE INITIAL CONDITIONS ON STATES OF D(Z)
C
      DO 699 INAG=1,3
      E1(INAG)=0.0
      E2(INAG)=0.0
      E1P(INAG)=0.0
      E2P(INAG)=0.0
      E1PP(INAG)=0.0
      699 E2PP(INAG)=0.0
      GC=1.0
      10 FORMAT(3(5X,E15.8))
      11 FORMAT(4(5X,E15.8))

```

```

50 CONTINUE
  READ(5,10)ALO,AL1,B0
  READ(5,10)B1,B2,B3
  READ(5,10)B4,B5,B6
  READ(5,11)B7,C1,C2,FC
  READ(5,11)ALCG, AIXX, AIE, TIME
  READ(5,11)ASE, AK3, AK4, P
  READ(5,11)AK7, AM, RP, V
  READ(5,11)WB(1), WB(2), WB(3), WB(4)
  READ(5,11)ZETA(1), ZETA(2), ZETA(3), ZETA(4)
  READ(5,11)GM(1), GM(2), GM(3), GM(4)
  READ(5,11)YB(1), YB(2), YB(3), YB(4)
  READ(5,11)YPB(1), YPB(2), YPB(3), YPB(4)
  READ(5,11)YPD(1), YPD(2), YPD(3), YPD(4)
  DO 9 I=1,4
    9 WB(I)=WB(I)*2.0*3.1415927
33 FORMAT(1H1,4X,10HINPUT DATA,/)
  AK3 = AK3*57.29578
  AK4 = AK4*57.29578
  AK7 = AK7*57.29578
  PRINT 33
  PRINT 10, ALO, AL1, B0
  PRINT 10, B1, B2, B3
  PRINT 10, B4, B5, B6
  PRINT 11, B7, C1, C2, FC
  PRINT 11, ALCG, AIXX, AIE, TIME
  PRINT 11, ASE, AK3, AK4, P
  PRINT 11, AK7, AM, RP, V
  PRINT 11, WB(1), WB(2), WB(3), WB(4)
  PRINT 11, ZETA(1), ZETA(2), ZETA(3), ZETA(4)
  PRINT 11, GM(1), GM(2), GM(3), GM(4)
  PRINT 11, YB(1), YB(2), YB(3), YB(4)
  PRINT 11, YPB(1), YPB(2), YPB(3), YPB(4)
  PRINT 11, YPD(1), YPD(2), YPD(3), YPD(4)
  ITER=25
  PRINT 17,ITER
17 FORMAT(1H1,2HN=I3,/)
  PRINT 26,TIME
26 FORMAT(15X,12HFLIGHT TIME=F5.1,1X,7HSECONDS,/)

C
C   N IS THE ORDER OF THE SYSTEM  A  MATRIX
C
  N=14

C
C   T IS THE SAMPLING PERIOD
C
  T=.04

C
C   DEFINE ELEMENTS OF  A  MATRIX
C
  DO 1 I=1,N

```

```

DO 1 J=1,N
1 A(I,J)=0.0
  W1=34.48
  ZETA1=0.434
  W2=84.09
  ZETA2=0.594
  A(1,1)=-2.*ZETA1*W1
  A(1,2)=-W1**2
  DO 2 I=2,N,2
    I1=I-1
  2 A(I,I1)=1.0
    A(3,2)=W1**2
    A(3,3)=-2.*ZETA2*W2
    A(3,4)=-W2**2
    A(5,2)=(ALCG*ASE+AIE)*W2**2*(-W1**2)/AIXX
    A(5,3)=(ALCG*ASE+AIE)*W2**2*2.*ZETA2*W2/AIXX
    A(5,4)=(ALCG*ASE+AIE)*W2**2*W2**2/AIXX + W2**2*(-C2-
1 AK3*ASE/AIXX)
    A(5,6)=-C1
    A(5,8)= -(FC*(ALCG*YPB(1)+YB(1)))/AIXX
    A(5,10)= -(FC*(ALCG*YPB(2)+YB(2)))/AIXX
    A(5,12)= -(FC*(ALCG*YPB(3)+YB(3)))/AIXX
    A(5,14)= -(FC*(ALCG*YPB(4)+YB(4)))/AIXX
    A(7,2)=(ASE*YB(1)-AIE*YPB(1))*W2**2*W1**2/GM(1)
    A(7,3)=(ASE*YB(1)-AIE*YPB(1))*W2**2*(-2.*ZETA2*W2)/
2 GM(1)+W2**2*RP*YB(1)
    A(7,4)=(ASE*YB(1)-AIE*YPB(1))*W2**2*(-W2**2)/
    A(7,7)=-2.*ZETA(1)*WB(1)
    A(7,8)=-WB(1)**2
    A(9,2)=W2**2*(ASE*YB(2)-AIE*YPB(2))*W1**2/GM(2)
    A(9,3)=W2**2*(ASE*YB(2)-AIE*YPB(2))*(-2.*ZETA2*W2)/
1 GM(2)
    A(9,4)=W2**2*(ASE*YB(2)-AIE*YPB(2))*(-W2**2)/GM(2) +
1 W2**2*RP*YB(2)/GM(2)
    A(9,9)=-2.*ZETA(2)*WB(2)
    A(9,10)=-WB(2)**2
    A(11,2)=W2**2*(ASE*YB(3)-AIE*YPB(3))/GM(3)*W1**2
    A(11,3)=W2**2*(ASE*YB(3)-AIE*YPB(3))/GM(3)*(-2.*ZETA2*
1 W2)
    A(11,4)=W2**2*(ASE*YB(3)-AIE*YPB(3))/GM(3)*(-W2**2) +
1 W2**2*RP*YB(3)/GM(3)
    A(11,11)=-2.*ZETA(3)*WB(3)
    A(11,12)=-WB(3)**2
    A(13,2)=W2**2*(ASE*YB(4)-AIE*YPB(4))/GM(4)*W1**2
    A(13,3)=W2**2*(ASE*YB(4)-AIE*YPB(4))/GM(4)*(-2.*ZETA2*
1 W2)
    A(13,4)=W2**2*(ASE*YB(4)-AIE*YPB(4))/GM(4)*(-W2**2) +
1 W2**2*RP*YB(4)/GM(4)
    A(13,13)=-2.*ZETA(4)*WB(4)
    A(13,14)=-WB(4)**2

```



```

C      COMPUTE PHI(T), THE STATE TRANSITION MATRIX
C
C      ITER=NUMBER OF TERMS USED IN TAYLOR SERIES EXPANSION
C      TO CALCULATE PHI(T)
C
      DO 3 I=1,N
      DO 3 J=1,N
      AN(I,J,1)=A(I,J)*T
      B(I,J)=A(I,J)*T
3     C(I,J)=A(I,J)*T
      DO 4 LL=2,ITER
      DO 5 I=1,N
      DO 5 J=1,N
5     A(I,J)=C(I,J)/FLOAT(LL)
      CALL MATMUL(A,B,N,C)
      DO 6 I=1,N
      DO 6 J=1,N
6     AN(I,J,LL)=C(I,J)
4     CONTINUE
      DO 7 I=1,N
      DO 7 J=1,N
7     AI(I,J)=0.0
      DO 8 I=1,N
8     AI(I,1)=1.0
      DO 12 I=1,N
      DO 12 J=1,N
12    AT(I,J)=AI(I,J)
      DO 15 I=1,N
      DO 15 J=1,N
      DO 15 LL=1,ITER
15    AT(I,J)=AT(I,J)+AN(I,J,LL)
      DO 13 I=1,N
      DO 13 J=1,N
13    PRINT 14,I,J,AT(I,J)
14    FORMAT(5X,3HAT(,I2,1H,,I2,2H)=E20.8)
C
C      COMPUTE THE INTEGRAL OF THE STATE TRANSITION MATRIX
C
      DO 18 I=1,N
      DO 18 J=1,N
      AIN(I,J,1)=AN(I,J,1)*T/2.
      AI(I,J)=AI(I,J)*T
      B(I,J)=AN(I,J,1)
18    C(I,J)=AIN(I,J,1)
      DO 20 LI=2,ITER
      LIL=LI+1
      DO 19 I=1,N
      DO 19 J=1,N
19    A(I,J)=C(I,J)/FLOAT(LIL)
      CALL MATMUL(A,B,N,C)
      DO 21 I=1,N

```

```

      DO 21 J=1,N
21  AIN(I,J,LI)=C(I,J)
20  CONTINUE
      DO 22 I=1,N
      DO 22 J=1,N
22  AANT(I,J)=AI(I,J)
      DO 23 I=1,N
      DO 23 J=1,N
      DO 23 LI=1,ITER
23  AANT(I,J)=AANT(I,J)+AIN(I,J,LI)
      DO 24 I=1,N
      DO 24 J=1,N
24  PRINT 25,I,J,AANT(I,J)
25  FORMAT(10X,5HAANT(,I2,1H,,I2,2H)=E20.8)

C
C      BD(I) IS THE PRODUCT OF THE INTEGRATED PHI(T) MATRIX
C      AND THE B MATRIX
C
      DO 27 I=1,N
27  BD(I)=0.0
      DO 28 I=1,N
28  BD(I)=AANT(I,1)
      PRINT 49
49  FORMAT(1H1,6X,4HPHID,8X,5HE1(1),8X,5HE2(1),7X,6HE1P(
1 1),7X,6HE2P(1),6X,7HE1PP(1),6X,7HE2PP(1))
C      DEFINE INITIAL CONDITIONS FOR THE SYSTEM STATES
C
      DO 29 I=1,N
29  X(I)=0.0
      X(6)=10.0
C      DEFINE A DO-LOOP TO UPDATE THE SYSTEM STATES AND
C      PRINT THE OUTPUT STATES DESIRED AT EACH SAMPLING
C      INSTANT
      JJ=1
      MM=1
      DO 999 M=1,2001
      KK=M-1
      TIME=T*FLOAT(KK)
      DO 31 I=1,N
31  X2(I)=0.0
      DO 32 I=1,N
      DO 32 K=1,N
32  X2(I)=X2(I)+AT(I,K)*X(K)
      PHID=X(6)+YPD(1)*X(8)+YPD(2)*X(10)+YPD(3)*X(12)+
1  YPD(4)*X(14)
      CSBM=YPD(2)*X(10)
      CONTRL(M) = PHID*GC
      CALL DIGCOM(CONTRL(M),BETAC)
888 PRINT 35,PHID,E1(1),E2(1),E1P(1),E2P(1),E1PP(1),E2PP(1),ERR
35  FORMAT(1H , 1P10E13.4)
      IF (M.NE.JJ) GO TO 777
      JJ=10+JJ

```

```
      X1(MM)=TIME
      Y2(MM)=E1(1)
      Y1(MM)=PHID
      MM=MM+1
777 DO 34 I=1,N
      34 X2(I)=X2(I)+BD(I)*BETAC
      DO 36 I=1,N
      36 X(I)=X2(I)
999 CONTINUE
      CALL PLOT(TX1,X1,TY1,Y1,NN)
      CALL PLOT(TX1,X1,TY2,Y2,NN)
      GO TO 50
16 STOP
END
```

COMPENSATOR SUBROUTINE

```

C
C
C   THE COEFFICIENTS ARE ENTERED INTO THIS PROGRAM AS
C   FRACTIONS, SINCE THEY HAVE BEEN QUANTIZED.  FOR EX.
C   B1 MUST BE WRITTEN IN THE FORM, N/65536.=N/2**16,
C   WHERE N IS AN INTEGER.
C
C
$IBFTC DIGCOM
      SUBROUTINE DIGCOM(A,B)
      COMMON/COM1/E1(3),E2(3),E1P(3),E2P(3),E1PP(3),E2PP(3)
C   COEFFICIENTS AND GAIN FOR FIFTH-ORDER COMPENSATION
C   FUNCTION
      A0=1.0
      A1=-3530./2048.
      A2=1877./2048.
      B1=-90289./65536.
      B2=34839./65536.
      A1P=-128502./65536.
      A2P=63017./65536.
      B1P=-127717./65536.
      B2P=62341./65536.
      A1PP=-65274./65536.
      A2PP=0.0
      B1PP=-65405./65536.
      B2PP=0.0
      AG=1.0
      E1(1)=A*255./15.
      AX=1.0
      BX=255.0
      CALL ROUND (E1(1),ERR,AX,BX)
      E2(1)=A0*E1(1)+A1*E1(2)+A2*E1(3)-B1*E2(2)-B2*E2(3)
      E1P(1)=E2(1)
      AX=1.0
      BX=512./AX
      CALL ROUND (E1P(1),ERR,AX,BX)
      E2P(1)=      E1P(1)+A1P*E1P(2)+A2P*E1P(3)-B1P*E2P(2)-
1 B2P*E2P(3)
      E1PP(1)=E2P(1)
      AX=32.
      BX=65535./AX
      CALL ROUND (E1PP(1),ERR,AX,BX)
      E2PP(1) =      E1PP(1)+A1PP*E1PP(2)+A2PP*E1PP(3)-B1PP*
1 E2PP(2)
J-B2PP*E2PP(3)
      EODA=E2PP(1)
      AX=32.
      BX=16384./AX
      CALL ROUND (E2(1),ERR,AX,BX)
C   TRUNCATION OF E2P(1) FOR FIFTH-ORDER COMPENSATOR
      AX=32.

```

```

BX=65536./AX
CALL ROUND (E2P(1),ERR,AX,BX)
C
C TRUNCATION OF E2P(1) FOR SIXTH-ORDER COMPENSATOR
AX=1.0
BX=65536./AX
CALL ROUND (E2P(1),ERR,AX,BX)
C
AX=32.
BX=16384./AX
CALL ROUND (E2PP(1),ERR,AX,BX)
AX=8.0
BX=2047./AX
CALL ROUND (EODA,ERR,AX,BX)
B=EODA*15./255.*AG
DO 1 J=1,2
  I=3-J
  E1(I+1)=E1(I)
  E2(I+1)=E2(I)
  E1P(I+1)=E1P(I)
  E2P(I+1)=E2P(I)
  E1PP(I+1)=E1PP(I)
1 E2PP(I+1)=E2PP(I)
RETURN
END

```

C THE FOLLOWING SUBROUTINE PLOTS THE VALUES OF PHID AND
 C E1(1) VERSUS TIME

```

C
$IBFTC P PLOT
      SUBROUTINE P PLOT(XTITLE,XTABLE,YTITLE,YTABLE,NTABLE)
1     XTAB(330),
      DIMENSION YTABLE(NTABLE),XTABLE(NTABLE),YTAB(330),
1     TEMPY(330),TEMPX(330),LLY(330),LY(330),LX(330),
1     PT(111)
      DIMENSION YTITLE(8),XTITLE(8),TITL(2)
2     YDN(6),XAC(7)
      DATA          PT/111*1H /,H6/1H /,HH6/1H*/,HHH6/
2     1H./,HHI/1HI/,
XHHM/1H-/ ,KASE/0/
      WRITE(6,205)
205    FORMAT(10H$NOHEADER)
      KASE=KASE+1
      IPTM=0
      KPLOT=0
      KAX=0
      NTAB=NTABLE
      YMIN=YTABLE(1)
      DO 2 I=2,NTABLE
2     IF(YTABLE(I).LT.YMIN) YMIN=YTABLE(I)
      YMAX=YTABLE(1)
      DO 3 I=2,NTABLE
3     IF(YTABLE(I).GT.YMAX) YMAX=YTABLE(I)
      ITAB=1
      NTAB=NTABLE
      IF(NTABLE.GT.110) NTAB=110
      XMIN=XTABLE(1)
      XMAX=XTABLE(NTAB)
4     KPLOT=KPLOT+1
      IM=1
      DO 10 I=ITAB,NTAB
      KEEP=ITAB
      KTAB=ITAB+1
      DO 5 J=KTAB,NTAB
5     IF(YTABLE(J).GT.YTABLE(KEEP)) KEEP=J
      YTAB(I)=YTABLE(KEEP)
      XTAB(I)=XTABLE(KEEP)
      YTABLE(KEEP)=-1.E20
10    CONTINUE
      YSCALE=(YMAX-YMIN)/50.
      XSCALE=(XMAX-XMIN)/110.
      YDN(1)=YMAX
      DO 60 I=2,5
60    YDN(I)=YDN(I-1)-YSCALE*11.0
      YDN(6)=YMIN
      XAC(1)=XMIN
      XAC(7)=XMAX
      DO 61 I=2,6

```

```

61  XAC(I)=XAC(I-1)+XSCALE*20.0
    DO 20 I=ITAB,NTAB
      TEMPY(I)=50.*((YTAB(I)-YMIN)/(YMAX-YMIN))+.5
      TEMPX(I)=110.*((XTAB(I)-XMIN)/(XMAX-XMIN))+1.5
      LLY(I)=TEMPY(I)
      LY(I)=51-LLY(I)
      LX(I)=TEMPX(I)
      IF(YTAB(I).EQ.0.) KAX=LY(I)
      IF((YTAB(I).GT.0.).AND.(YTAB(I+1).LT.0.)) GO TO 19
    GO TO 20
19  AXIS=TEMPY(I)-50.*YTAB(I)/(YMAX-YMIN)
    KAXIS=AXIS
    KAX=51-KAXIS
20  CONTINUE
    WRITE(6,200) KASE,KPLOT,YSCALE,XSCALE
200  FORMAT(1H1,4HCASE,I3,8H....PLOT,I2, /16H SCALE....Y-
1  AXIS,F7.3,10H....X-AXIS,F7.3,/)
    X....X-AXIS,F7.3,/)
    DO 24 I=1,111
24  PT(I)=H6
    LINE=0
    LL=0
25  LINE=LINE+1
    PT(1)=HHH6
    IF(LINE.EQ.1) PT(1)=HHM
    LM1=LINE-1
    IF(LM1/10*10.EQ.LM1) PT(1)=HHM
123  IF(LINE.NE.KAX) GO TO 23
    DO 22 I=1,111,2
22  PT(I)=HHH6
    DO 122 I=21,101,20
122  PT(I)=HHI
23  DO 26 I=ITAB,NTAB
    IF(LY(I).NE.LINE) GO TO 26
    LXI=LX(I)
    PT(LXI)=HH6
26  CONTINUE
    TITL(1)=H6
    TITL(2)=H6
    DO 150 MM=24,33,3
    IF(LINE.EQ.MM) GO TO 51
    GO TO 150
51  LL=LL+1
    TITL(1)=YTITLE(LL)
    LL=LL+1
    TITL(2)=YTITLE(LL)
150  CONTINUE
    IF(LINE.EQ.1.AND.PT(1).EQ.HHH6) PT(1)=HHM
    IF(LINE.EQ.1) GO TO 27
    IF(LINE.EQ.51) GO TO 28
    LM1=LINE-1

```

```

        IF(LM1/10*10.NE.LM1) GO TO 31
        IM=IM+1
        WRITE(6,112) YDN(IM),PT
112    FORMAT(9X,F7.2,2X,111A1)
        GO TO 29
31    CONTINUE
        WRITE(6,100) TITL,PT
        GO TO 29
27    WRITE(6,105) YMAX,PT
        GO TO 29
28    WRITE(6,105) YMIN,PT
29    CONTINUE
        DO 30 I=1,111
30    PT(I)=H6
        IF(LINE.LT.50) GO TO 25
        IF(LINE.GT.50) GO TO 40
        DO 35 I=1,111,2
35    PT(I)=HHH6
        DO 135 I=21,101,20
135    PT(I)=HHI
        GO TO 25
40    CONTINUE
        WRITE(6,110)XAC
110    FORMAT(1H0,15X,F8.3,10X,F7.2,4(12X,F8.2),F10.3)
        WRITE(6,111) XTITLE
111    FORMAT(1H0,59X,8A6)
        NTABL=NTABL-110
        IF(NTABL.LE.0) GO TO 50
        XMAX=2.*XMAX-XMIN
        ITAB=ITAB+110
        XMIN=XTABLE(ITAB)
        IF(NTABL.GE.110) GO TO 45
        NTAB=NTABLE
        GO TO 4
45    NTAB=NTAB+110
        XMAX=XTABLE(NTAB)
        GO TO 4
50    CONTINUE
100    FORMAT(3X,2A6,3X,111A1)
101    FORMAT(12F6.1)
102    FORMAT(1H1,4HCASE,I3,8H....PLOT,I2,/)
104    FORMAT(12I6)
105    FORMAT(1H ,8X,F8.3,1X,111A1)
106    FORMAT(1H0,15X,F8.3,99X,F8.3,/,60X,8A6)
107    FORMAT(///,1X,15HSCALE    Y-AXIS,E9.3,10H    X-AXIS,
2 E9.3)
        RETURN

```



```

$IBFTC ROUND
      SUBROUTINE ROUND(A,B,AN,BN)
      X=ABS(A)
      S=A/X
      IX=X*AN
      XQ=IX
      XQ=XQ/AN
      IF(XQ-BN) 1,2,2
1  A=S*XQ
  B=S*(X-XQ)
  RETURN
2  A=S*BN
  B=S*(X-BN)
  RETURN
  END
$IBFTC MATMUL
      SUBROUTINE MATMUL(A,B,N,C)
      DIMENSION A(N,N),B(N,N),C(N,N)
C  CALCULATE C(I,J) COEFFICIENTS
      DO 3 I=1,N
      DO 3 J=1,N
      C(I,J)=0.0
      DO 4 K=1,N
4  C(I,J)=C(I,J)+A(I,K)*B(K,J)
3  CONTINUE
      RETURN
      END

```

```

$IBMAP TRAP      NODECK,NOLIST,NOREF
        ENTRY    TRAP
        AXT      **,4
TRAP    TRA      **
        SXA      TRAP-1,4
        CLA      8
        STA      RESET+1
        CLA      FIX
        TSX      S.SCCR,4
        STO      8
        TRA      TRAP-1
RESET   PLT      0
        TRA      **
        CLA      0
        ARS      20
        LBT
        TRA      *+2
        TRA*     RESET+1
        SXA      OUT,4
        TSX      S.WRIT,4
        PZE      3,,MES
OUT     AXT      **,4
        ZAC
        LRS      35
        TRA*     0
FIX     TRA      RESET
MES     BCI      3, **** UNDERFLOW
        END
$ENTRY
C
C      DATA CARDS
C
$IBSYS

```

APPENDIX B

FORTRAN SOURCE PRIGRAM FOR DETERMINING THE POSITIONING OF
THE COEFFICIENT SWITCHES

SOURCE DECK

```

      INTEGER IS(18)
      REAL A(7),COEF(4)
      DATA A/1HA,1HB,1H1,1H2,1H0,1HP,1H /
      WRITE (6,100)
100  FORMAT(1H1,9X,1HS,3X,1H1,3X,1H2,3X,1H3,3X,1H4,3X,
11H5,3X,1H6,3X,1H7,3X,1H8,3X,1H9,2X,2H10,2X,2H11,
12X,2H12,2X,2H13,2X,2H14,2X,2H15,2X,2H16,2X,2H17,2X,2H18)
      WRITE (7,105)
105  FORMAT(5X,1HS,3H 1,3H 2,3H 3,3H 4,3H 5,3H 6,
1 3H 7,3H 8,3H 9,3H 10,3H 11,3H 12,3H 13,3H 14,
2 3H 15,3H 16,3H 17,3H 18)
      4  READ(5,101)COEF,VALUE
      IF(COEF(1).EQ.A(7))GO TO 5
101  FORMAT (4A1,F11.0)
      ISO=0
      IF(VALUE.LT.0.0)ISO=1
      VALUE=ABS(VALUE)
      IVAL=VALUE
      IS(1)=0
      IF(IVAL.GT.1)IS(1)=1
      IVA=IVAL-IS(1)*2
      IS(2)=0
      IF(IVA.EQ.1)IS(2)=1
      DVAL=VALUE-FLOAT(IVAL)
      DO 1 I=3,18
      IS(I)=ABS(DVAL*2.)
      DVAL=DVAL*2.-FLOAT(IS(I))
1  CONTINUE
      IF(COEF(1).EQ.A(1).AND.COEF(2).EQ.A(5))GO TO 2
      IF(COEF(1).EQ.A(1).AND.COEF(3).NE.A(6))GO TO 3
      WRITE(6,102)COEF,ISO,(IS(I),I=2,18)
      WRITE(7,107)COEF,ISO,(IS(I),I=2,18)
107  FORMAT(4A1,I2,3X,17I3)
      GO TO 4
      2  WRITE(6,103)COEF,ISO,(IS(I),I=2,7)
103  FORMAT(1H0,3X,4A1,2X,I1,4X,6I4)

```

```

102  FORMAT(1H0,3X,4A1,2X,I1,4X,17I4)
      WRITE(7,108)COEF,ISO,(IS(I),I=2,7)
108  FORMAT(4A1,I2,3X,6I3)
      GO TO 4
      3  WRITE(6,104)COEF,ISO,(IS(I),I=1,12)
104  FORMAT(1H0,3X,4A1,2X,I1,12I4)
      WRITE(7,109)COEF,ISO,(IS(I),I=1,12)
109  FORMAT(4A1,I2,12I3)
      GO TO 4

```

C THE NAME OF THE COEFFICIENT IS ENTERED IN COLUMNS 1-4,
C AND THE VALUE OF THE COEFFICIENT IS ENTERED IN
C COLUMNS 6-15. THE SWITCH POSITIONS FOR THE FIFTH-ORDER
C COMPENSATOR ARE GIVEN BELOW.
C

5 STOP
END

SENTRY

```
A0      1.0
A1      -1.7237237
A2      .9162799
B1      -1.3777001
B2      .5316022
A1P     -1.9607921
A2P     .9615609
B1P     -1.94881
B2P     .9512492
A1PP    -.9960079
A2PP    0.0
B1PP    -.9980019
B2PP    0.0
```

\$IBSYS

C THE POSITIONING OF THE COEFFICIENT SWITCHES FOR THE
C FIFTH-ORDER COMPENSATION FUNCTION ARE GIVEN BELOW.

[illegible]